

**Universidad Nacional Autónoma de
México**

Facultad de Ingeniería

Profesor: Gunnar Eyal Wolf Iszaevich

Materia: Sistemas Operativos

Grupo: 06

Alumnos:

Jesús Abner Domínguez Chávez

Saida Mayela Sánchez Calvillo

**Proyecto 01. Microsistema de archivos
multihilos**

Fecha de Entrega: 19 - Mayo - 2024

REQUISITOS DEL PROGRAMA

Desarrollar un programa que pueda obtener, crear y modificar información en el micro-sistema-de-archivos de la Facultad de Ingeniería, FiUnamFS.

Siguiendo la siguiente especificación:

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

El programa debe contar, por lo menos, dos hilos de ejecución, operando concurrentemente, y que se comuniquen su estado mediante mecanismos de sincronización.

EXPLICACIÓN DEL PROGRAMA

```
import os
import struct
from struct import *
import threading
```

Lo primordial son los módulos incluidos en el código. Se utiliza “os” para poder hacer uso de funciones que nos permitan manipular archivos que involucren a la computadora que corre el programa. El módulo struct es vital para la resolución de éste problema, ya que, nos sirve para poder convertir cadenas de bytes a números y cadenas. Por último, el módulo threading va a ayudarnos a tener funciones que nos sirvan para manejar hilos y poder hacer nuestro código concurrente.

```
#Se muestran los datos del superbloque
FS.seek(0)
nombre_FS = FS.read(8).decode('ascii')
FS.seek(10)
version = FS.read(10).decode('ascii')
etiqueta_volumen = FS.read(20).decode('ascii')
totalsuperbloque = FS.read(54)
FS.seek(40)
tamaño_cluster = unpack("<I",FS.read(4))[0]
num_cluster_dir = unpack("<I", totalsuperbloque[45:49])[0]
num_cluster_uni = unpack("<I", totalsuperbloque[50:54])[0]
```

En la primera parte del código se comienza con la lectura del primer cluster del sistema de archivo dentro de la imagen. Utilizamos el método seek() para saltar a ciertos bytes del cluster y utilizar el método read() para leer un rango específico de bytes en donde se encuentra diversa información, por ejemplo, el nombre del sistema de archivos, versión, etiqueta de volumen, etc. Para cuestiones de los clusters, se usa el método unpack() con formato “<I” para extraer la información del superbloque.

```

# Función para leer los datos del archivo enteros
def leerDatos(inicio, tamano):
    global FS
    FS.seek(inicio)
    return FS.read(tamano)

# Función para leer los datos ASCII del FS
def leerASCII(inicio, tamano):
    global FS
    FS.seek(inicio)
    datos = FS.read(tamano)
    return datos.decode("ascii")

# Para convertir los datos de ASCII
def datoUnpack(inicio, tamano):
    global FS
    FS.seek(inicio)
    dato = FS.read(tamano)
    return struct.unpack('<i', dato)[0]

# Para escribir datos al sistema de archivos
def datosPack(inicio, dato):
    global FS
    FS.seek(inicio)
    dato = struct.pack('<i', dato)
    return FS.write(dato)

# Función para leer los datos del archivo en ASCII
def leerArchivo(posicion):
    inicio = 1024 + (posicion * 64)
    nombre = leerASCII(inicio + 1, 14)
    if nombre.strip('.') != "":
        tamano = datoUnpack(inicio + 16, 4)
        clusterInicial = datoUnpack(inicio + 20, 4)
        return archivo(nombre, tamano, clusterInicial)
    return None

```

Se cuentan con algunas funciones auxiliares que nos facilitaran algunas acciones que se necesitan varias veces durante toda la ejecución del programa.

Como el nombre de cada función lo menciona, podemos leer datos binarios, leerlos y decodificarlos, convertir un dato con el formato “<I” usando unpack() o viceversa usando formato “<I” y pack() podemos convertir nuestro dato a un archivo que pueda guardas en el sistema de archivos. Por ultimo, tenemos una función que nos permite leer los datos de un archivo perteneciente del sistema de archivos y convertirlo en ASCII para poderse leer de manera normal.

La función verificarArchivo, hace un recorrido simple a una de lista de listas en la cual cada lista dentro del arreglo es la información de cada archivo que contiene la imagen FiUnamFS.

Para la función que nos permite exportar un archivo del sistema de archivos, se apoya de la función anteriormente mencionada y se le ingresan datos como argumentos obtenidos de algunas preguntas al usuario para obtener detalles sobre el archivo a exportar. Con esto obtenido, usamos funciones del modulo “os” para crear y escribir el archivo que tendrá la información deseada por el usuario.

```

def verificarArchivo(nombreCopia):
    global archivos
    with archivo_lock:
        for i in archivos:
            if i.nombre.strip() == nombreCopia.strip():
                return archivos.index(i), True
        return -1, False

# Función para copiar un archivo del sistema de archivos a la computadora
def exportar(nombreCopia, rutaNueva):
    # Verificamos si el archivo que se quiere copiar existe en nuestro directorio
    indexArchivo, validacion = verificarArchivo(nombreCopia)
    if not validacion:
        print("El archivo no existe")
        return

    # Archivo que se quiere copiar
    archivoC = archivos[indexArchivo]

    print(f"Tamaño del archivo a copiar: {archivoC.tamano}")

    # Crear el archivo en la ruta especificada
    if os.path.exists(rutaNueva):
        if os.path.isfile(os.path.join(rutaNueva, nombreCopia)):
            rutaArchivoDestino = os.path.join(rutaNueva, "copia de " + nombreCopia)
        else:
            rutaArchivoDestino = os.path.join(rutaNueva, nombreCopia)

        with open(rutaArchivoDestino, "wb") as destino:
            inicio_lectura = archivoC.clusterInicial * tamanoClusters
            FS.seek(inicio_lectura)
            datos_archivo = FS.read(archivoC.tamano)
            destino.write(datos_archivo)

        print("Archivo copiado con éxito")
    else:
        print("La ruta especificada no existe")

```

```
def importar(rutaArchivo):
    global FS, tabla_asignacion, archivos, tamanoClusters

    if not os.path.exists(rutaArchivo):
        print("El archivo no existe.")
        return

    nombreArchivo = os.path.basename(rutaArchivo)

    # validar si el archivo ya existe en el sistema de archivos
    if verificarArchivo(nombreArchivo)[1]:
        print("El archivo ya existe en el sistema de archivos.")
        return

    tamanoArchivo = os.path.getsize(rutaArchivo)
    espacioDisponible = verEspacioDisponible(tamanoArchivo)
    print(f"Espacio disponible: {espacioDisponible}")

    if espacioDisponible == -1:
        print("No hay suficiente espacio en el sistema de archivos para copiar el archivo.")
        return

    with open(rutaArchivo, "rb") as archivoComputadora:
        contenido = archivoComputadora.read()

        # Escribir en el espacio disponible encontrado
        inicio_escritura = espacioDisponible
        print("Inicio escritura: ", inicio_escritura)
        FS.seek(inicio_escritura)
        FS.write(contenido)

    nuevoArchivo = archivo(nombreArchivo, tamanoArchivo, espacioDisponible)
    print("Nombre del archivo: ", nuevoArchivo.nombre)
    with archivo_lock:
        archivos.append(nuevoArchivo)
        agregar(nuevoArchivo)
    print("Archivo copiado con éxito al sistema de archivos")
```

En la función importar, se comienza haciendo ciertas verificaciones necesarias para evitar problemas al querer importar un archivo que ya exista en el sistema de archivos o por el otro lado sea un archivo inexistente, todo esto a través de la función antes explicada “verificarArchivo”.

Para determinar el tamaño y el espacio que se usará para guardar el archivo se utiliza una función del módulo “os” para encontrar el tamaño del archivo y para el espacio disponible en el sistema de archivos se usa una función

que se explicará a continuación.

Cuando todo lo anterior haya sido ejecutado con éxito, se creará un nuevo archivo en el sistema de archivos, se escribirá su contenido que fue leído mediante la función read() y para finalizar, se agregará toda la información del archivo al arreglo que contiene toda la información de todos los archivos del filesystem.

Para poder encontrar un espacio disponible para guardar el archivo importado por el usuario, se hará mediante la verificación de los clusters del filesystem a través de la lista de archivos. Se

```
def verEspacioDisponible(tamanoArchivo):
    global FS, archivos, tamanoClusters, num_cluster_dir

    with archivo_lock:
        # Conjunto de todos los clusters ocupados por archivos existentes
        clustersOcupados = set()

        # Tenemos que calcular los clusters ocupados en el estado actual
        for archivo in archivos:
            clusters = [archivo.clusterInicial + i for i in range((archivo.tamano + tamanoClusters - 1) // tamanoClusters)]
            clustersOcupados.update(clusters)

        # Busca un espacio libre
        espacioLibre = -1
        i = 0
        while i < num_cluster_dir:
            if i not in clustersOcupados and i * tamanoClusters > 54: # Excluye los primeros 54 bytes
                inicio = i * tamanoClusters
                FS.seek(inicio)
                # Verifica si el espacio libre encontrado es suficiente para el archivo que se va a escribir
                if (i + (tamanoArchivo + tamanoClusters - 1) // tamanoClusters) < num_cluster_dir:
                    return i # Devuelve el índice del espacio libre encontrado
                i += 1
        return -1 # Si no se encuentra ningún espacio libre, devuelve -1
```

recorrerá la lista de archivos marcando los clusters ocupados por estos mismos, hasta encontrar un cluster libre y por consiguiente verificando si el tamaño del archivo cabe en el cluster libre encontrado, si esto es verdadero, se retornará el índice del espacio libre.

```

def eliminar(nombreArchivo):
    global archivos
    global tamañoClusters
    global FS

    with archivo_lock:
        # Verificamos si el archivo que se quiere borrar existe en nuestro directorio
        indexArchivo, validacion = verificarArchivo(nombreArchivo)
        if not validacion:
            print("El archivo no existe")
            return

        # Archivo que se quiere borrar
        archivoBorrar = archivos[indexArchivo]

        # Eliminamos el archivo del directorio
        archivos.pop(indexArchivo)

        # Marcar la entrada en el directorio como eliminada
        FS.seek(1024 + indexArchivo * 64)
        FS.write(b'\x00')

        # Marcamos los clusters correspondientes como libres
        FS.seek(archivoBorrar.clusterInicial * tamañoClusters)
        FS.write(b'\x00' * archivoBorrar.tamaño)

        # Recargar la lista de archivos del directorio
        archivos = []
        listarDirectorio()

    print("Archivo eliminado con éxito")

```

Como última función, tenemos la función eliminar, la cual como argumento recibe el nombre del archivo que a través de la función “verificarArchivo” observaremos si realmente existe dentro de nuestro sistema de archivos, cuando esto sea verificado, sacaremos de la lista de listas que contiene toda la información de los archivos, el archivo que haya escogido el usuario. Después se marcará la entrada como eliminada escribiendo en el sistema de archivos b'\x00' además de que se debe de marcar el cluster que le correspondía a

ese archivo como libre, para hacer esto con seek() ubicaremos el puntero en el índice del cluster correspondiente y escribiremos de igual forma b'\x00' para liberar el cluster anteriormente ocupado. Para finalizar, volvemos a cargar la lista de archivos con la eliminación realizada de manera correcta y a su vez notificando al usuario que se hizo correctamente la eliminación.

```

def menu():
    while True:
        print("\n1. Listar el contenido del directorio")
        print("2. Copiar archivo del sistema a la computadora")
        print("3. Copiar archivo de la computadora al sistema")
        print("4. Borrar archivo del sistema")
        print("5. Salir")
        try:
            opcion = int(input("Ingresa una opción: "))
            if opcion == 1:
                listarArchivos()
            elif opcion == 2:
                listarDirectorio()
                nombreCopia = input("Ingresa el nombre del archivo que deseas copiar (incluye la extensión): ")
                rutaCopiar = os.path.dirname(os.path.abspath(__file__))
                threading.Thread(target=exportar, args=(nombreCopia, rutaCopiar)).start()
            elif opcion == 3:
                rutaArchivo = input("Ingresa la ruta de donde deseas copiar el archivo (incluye el archivo con su extensión): ").replace("\\", "/")
                threading.Thread(target=importar, args=(rutaArchivo,)).start()
            elif opcion == 4:
                nombreArchivo = input("Ingresa el nombre del archivo que deseas borrar (incluye la extensión): ")
                threading.Thread(target=eliminar, args=(nombreArchivo,)).start()
            elif opcion == 5:
                break
            else:
                print("Opción inválida")
        except ValueError:
            print("Por favor, ingrese un número válido")

```

Dentro del menú tenemos todo lo anterior implementado de manera que el usuario pueda interactuar con nuestro programa de manera natural sin tener que meterse con el código o cosas extrañas. Se puede observar que las funciones no son llamadas de manera directa cada que se quieren usar, si no que se hacen a través de hilos, donde cada opción del menú es un hilo diferente y nuestro hilo principal por el cual el programa es ejecutado inicialmente es el menú.

Método de sincronización empleada

```
# Lock para acceso seguro a la lista de archivos
archivo_lock = threading.Lock()
```

Para este programa optamos por utilizar el método de sincronización de hilos llamado lock. Éste método de sincronización nos permite establecer un bloqueo para regular el acceso a un

recurso compartido, en este caso, la lista de archivos que contiene toda la información de los archivos que contiene el sistema de archivos.

Podemos observar que en distintas partes del programa, cada que se requiere acceder a la lista de archivos, las instrucciones están dentro de un “with archivo_lock”, instrucción que implícitamente está ejecutando los métodos .acquire y .release para bloquear y desbloquear el recurso compartido, esto para poder bloquear la lista hasta que se terminen de ejecutar las instrucciones y por consiguiente liberar el bloqueo para que cualquier otro hilo o proceso haga uso de este bloqueo y a su vez la lista.

En las imágenes se puede observar lo explicado anteriormente, como las instrucciones que requieren la lista, están dentro del “with archivo_lock”.

```
def listarDirectorio():
    global tamanoClusters
    global num_cluster_dir
    global tamanoDirectorio
    global archivos

    with archivo_lock:
        archivos = []

        print("\033[1m Nombre archivo\t\tTamaño \033[0m")

def verificarArchivo(nombreCopia):
    global archivos
    with archivo_lock:
        for i in archivos:

print("Nombre del archivo: ", nuevoArchivo.nombre)
with archivo_lock:
    archivos.append(nuevoArchivo)
    agregarAlDirectorio(nuevoArchivo)
    print("Archivo copiado con éxito al sistema de archivos")

with archivo_lock:
    # Conjunto de todos los clusters ocupados por archivos existentes
    clustersOcupados = set()

    for archivo in archivos:
        # Calcula todos los clusters ocupados por el archivo actual
        clusters = [archivo.clusterInicial + i for i in range((archivo.tamano - archivo.clusterInicial))]
        clustersOcupados.update(clusters)

with archivo_lock:
    # Verificamos si el archivo que se quiere borrar existe en la lista
    indexArchivo, validacion = verificarArchivo(nombreArchivo)
    if not validacion:
        print("El archivo no existe")
    return
```

EJEMPLO DE USO

NOTA: Es importante que al ejecutar el programa, se comience utilizando la opción 1 y después la opción 2, después de esto, se puede ejecutar cualquier opción sin ningún orden en específico

```
Nombre del sistema de archivos: FiUnamFS
Version: 24-2
Etiqueta del Volumen: Mi Sistema Favorito
Tamaño del cluster: 2048 bytes
Numero de cluster que mide el directorio: 4
Numero de cluster que mide la unidad completa: 720

1. Listar el contenido del directorio
2. Copiar archivo del sistema a la computadora
3. Copiar archivo de la computadora al sistema
4. Borrar archivo del sistema
5. Salir
Ingresa una opción: █
```

Ejecutando la opción número 1 se obtiene:




```
Tipo de archivo: -
Archivo: README.org
Fecha de creación del archivo: 2024 - 05 - 08 13 : 17 : 56
Fecha de modificación del archivo: 2024 - 05 - 08 13 : 17 : 56

Tipo de archivo: -
Archivo: logo.png
Fecha de creación del archivo: 2024 - 05 - 08 13 : 17 : 56
Fecha de modificación del archivo: 2024 - 05 - 08 13 : 17 : 56

Tipo de archivo: -
Archivo: mensaje.jpg
Fecha de creación del archivo: 2024 - 05 - 08 13 : 17 : 56
Fecha de modificación del archivo: 2024 - 05 - 08 13 : 17 : 56
```


Al escoger la opción 2, se copiará el archivo “logo.png” al directorio donde se encuentra nuestro programa

```
Ingresa una opción: 2
Nombre archivo      Tamaño
README.org          31222 bytes
logo.png             126423 bytes
mensaje.jpg          254484 bytes
Ingresa el nombre del archivo que deseas copiar (incluye la extensión): logo.png
Tamaño del archivo a copiar: 126423
```

Nombre	Fecha de modificación	Tipo	Tamaño
 fiunamfs.img	18/05/2024 12:27 p. m.	DAEMON.Tools.Lite	1,440 KB
 proyecto	20/05/2024 12:27 a. m.	Archivo de origen ...	14 KB
 logo	20/05/2024 12:29 a. m.	Archivo PNG	124 KB

Se copia el archivo “prueba.txt” de nuestra computadora al sistema de archivos

```
Ingresa una opción: 3
Ingresa la ruta de donde deseas copiar el archivo (incluye el archivo con su extensión): C:\Users\Mayela Calvillo\Desktop\prueba.txt

1. Listar el contenido del directorio
2. Copiar archivo del sistema a la computadora
3. Copiar archivo de la computadora al sistema
4. Borrar archivo del sistema
Espacio disponible: 1
5. Salir
Inicio escritura: 1
Nombre del archivo: prueba.txt
Ingresa una opción: Archivo copiado con éxito al sistema de archivos
1
Tipo de archivo:
Archivo: prueba.txt
Fecha de creación del archivo: - - : :
Fecha de modificación del archivo: - - : :
```

Aquí eliminamos mensaje.jpg de nuestro sistema de archivos

```
Ingresa una opción: 4
Ingresa el nombre del archivo que deseas borrar (incluye la extensión): mensaje.jpg

1. Listar el contenido del directorio
2. Copiar archivo del sistema a la computadora
3. Copiar archivo de la computadora al sistema
4. Borrar archivo del sistema
5. Salir
Nombre archivo      Tamaño
README.org          31222 bytes
logo.png             126423 bytes
Ingresa una opción: mensaje.jpg      254484 bytes
Archivo eliminado con éxito
```