Universidad Nacional Autónoma de México

Facultad de Ingeniería

División de Ingeniería Eléctrica

Sistemas Operativos

Profesor(a): Gunnar Wolf

Semestre 2024-2

Proyecto

Documentación

(Micro) Sistema de Archivos Multihilos

Integrantes

➢ Flores Melquiades Evelyn Jasmin - 319112186

➢ Vera Garmendia Miriam Marisol - 319239748

Grupo: __6___

Cd. Universitaria a 19 de mayo de 2024

Proyecto: (Micro) sistema de archivos multihilos

Introducción:

El presente proyecto deberá de darla resolución al siguiente problema planteado:

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

Dicho programa deberá de:

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
5) Contar por lo menos, dos hilos de ejecución, operando concurrentemente, y que se comuniquen su estado mediante mecanismos de sincronización.

Además, deberá de contar con las siguientes especificaciones:

- El sistema de archivos cabe en un diskette tradicional. Claro, no espero que tengan acceso al hardware, por lo que lo simularemos representándolo en un archivo de longitud fija, de 1440 Kilobytes
- Por simplicidad, en todas las estructuras de FlUnamFS, las cadenas de texto deben ser ASCII 8-bit (no requerimos que sean Unicode UTF-8).
- En las estructuras del disco, todos los enteros serán representados como valores de 32 bits, en formato little endian.
    - Esto es, por ejemplo: El valor 1354 se representará como los bytes (74, 5, 0, 0) (porque ((0 * 256 + 0) * 256 + 5) * 256 + 74 = 1354). El valor 1234567890 se representará como los bytes 210, 2, 150, 7 (porque ((7 * 256 + 150) * 256 + 2) * 256 + 210 = 1234567890.
    - Para hacer las conversiones desde o hacia este formato, les sugiero usar el formato <I con las funciones pack() y unpack(), disponibles en distintos lenguajes (Python: struct.pack(), Ruby: Array#pack(), Perl: pack(), PHP: pack(), hay varias biblitecas que pueden usar para Javascript, como node-jspack o bufferpack). Si van a usar C o C++, les sugiero revisar las funciones htons() y ntohs(). ¿Java? Si bien no forman parte del lenguaje, pueden encontrar varias implementaciones de htons() y ntohs() en la red.
- La superficie del disco se divide en sectores de 512 bytes. Cada cluster mide cuatro sectores.
- El pseudodispositivo no maneja tabla de particiones, sino que hospeda directamente un volumen en la totalidad de su espacio.
- FiUnamFS maneja únicamente un directorio plano, no se consideran subdirectorios.
- El primer cluster (#0) del pseudodispositivo es el superbloque. Este contiene información en los siguientes bytes:
    - **0–8:** Para identificación, el nombre del sistema de archivos. ¡Debes validar nunca modificar un sistema de archivos que no sea el correcto! Debe ser la cadena FiUnamFS.

- ❖ **10–14:** Versión de la implementación. Estamos implementando la versión 24-2 (ojo, es una cadena, no un número). Deben validar que el sistema de archivos a utilizar sea de esta versión, para evitar la corrupción de datos.
- ❖ **20–35:** Etiqueta del volumen
- ❖ **40–44:** Tamaño del cluster en bytes
- ❖ **45–49:** Número de clusters que mide el directorio
- ❖ **50–54:** Número de clusters que mide la unidad completa

El resto del superbloque puede quedar vacío (o puedes sobrecargarlo con otros datos — No es importante para la implementación.

- El sistema de archivos es de asignación continua. Toda la información de los archivos está en el directorio.
- El directorio está ubicado en los clusters 1 a 4. Cada entrada del directorio mide 64 bytes, consistentes en:

❖ **0:** Tipo de archivo. Dado que el sistema de archivos actual no tiene soporte para directorios, dispositivos, pipes, ni otros archivos especiales, siempre será el carácter - (0x2d, 45). Cuando la entrada está vacía, se indica con el carácter / (0x2f, 47).

- ❖ **1–15:** Nombre del archivo
- ❖ **16–20:** Tamaño del archivo, en bytes
- ❖ **20–23:** Cluster inicial
- ❖ **24–37:** Hora y fecha de creación del archivo, especificando AAAAMMDDHHMMSS (p.ej. '20221108182600' para 2022-11-08 18:26:00)
- ❖ **38-51:** Hora y fecha de última modificación del archivo, especificando AAAAMMDDHHMMSS (p.ej. '20221109182600')
- ❖ **52–64:** Espacio no utilizado (¿reservado para expansión futura?)

- Las entradas no utilizadas del directorio se identifican porque en el campo de nombre llevan la cadena ###############.
  - Los nombres de archivos pueden componerse de cualquier carácter dentro del subconjunto ASCII de 7 bits (no acentuados, no Unicode, sólo el viejo y aburrido US-ASCII)
- Es un sistema de archivos plano — No maneja subdirectorios.
- Después del directorio, todo el espacio restante es espacio de datos.

Ambiente en el que se desarrolló

1) Principalmente para trabajar en conjunto se utilizó *REPLIT.com* la cual es una herramienta que cuenta con gran variedad de lenguajes a utilizar y permite el trabajo colaborativo entre varias personas al mismo tiempo.

```
lock-version = "2.0"
python-versions = ">=3.10.0,<3.12"
```

2) El segundo que se usó fue *spyder (Python 3.9)* el cual fue empleado mas que nada para trabajar en conjunto con GIT y así realizar los commits correspondiente al momento de ir desarrollando lo mas relevante del programa

NOTA IMPORTANTE: El lenguaje en el que se desarrolló el proyecto fue **PYTHON**

Desarrollo:

- **Importación de módulos:**
  - **import os:** Para interactuar con el sistema operativo.
  - **import struct:** Que permite interpretar cadenas de bytes como estructuras binarias.
  - **import threading:** Que proporciona funcionalidad de subprocesos para realizar múltiples tareas de forma simultánea.

```
import os #para interactuar con sistema operativo
import struct
import threading #Manejo de hilos
```

Algunas variables globales que utilizamos y que venían especificadas en el planteamiento del problema son:

```
TAMANO_CLUSTER = 512
TAM_ENTRADA = 64
# Cluster
DIRECTORIO_INICIO = TAMANO_CLUSTER
# 4 sectores
DIRECTORIO_TAMANO = 4 * TAMANO_CLUSTER
# Tomando en cuenta que el tamaño total es 1440KB
MAXIMO_CLUSTERS = 1440 // 4
#Semaforo para sincronizar la lectura de archivos
semaforo = threading.Semaphore(value=1)
#Sincronizar hilos
lock = threading.Lock()
```

Se crearán diferentes funciones, la primera será

- **ENTERO:** Lo que realizará esta función será que leerá un entero desde el archivo en una posición específica.

```
def ENTERO(DIRECTORIO_INICIO, TAM_ENTRADA):
    with open(fiunamfs_img, "rb") as f:
        f.seek(DIRECTORIO_INICIO)
        DATA = f.read(TAM_ENTRADA)
        E = struct.unpack("<I", DATA)[0]
        return E
```

- **CADENA:** Lo que realizará esta función será que leerá una cadena desde el archivo en una posición específica.

```
def CADENA(DIRECTORIO_INICIO, TAM_ENTRADA):
    with open(fiunamfs_img, "rb") as f:
        f.seek(DIRECTORIO_INICIO)
        CADENA = f.read(TAM_ENTRADA)
        string = CADENA.decode("ascii").rstrip()
        return string
```

❖ **NOTA:** A lo largo del código se utilizarán las siguientes funciones:

- ➢ **archivo.seek(DIRECTORIO_INICIO):** El cual mueve el puntero del archivo a la posición especificada.
- ➢ **f.read(TAM_ENTRADA):** El cual lee el número de bytes especificado desde la posición actual.
- ➢ **tipo_archivo = CADENA(DIRECTORIO_TAMANO + _ , *1):*** *Lee el tipo de archivo desde el cluster actual más el desplazamiento* '_' con longitud de 1 byte
- ➢ **nombre = CADENA(DIRECTORIO_TAMANO + _ + 1, 15):** Lee el nombre del archivo desde el cluster actual más el desplazamiento '_' más 1 con longitud 15 bytes
- ➢ **tam= leerEntero(DIRECTORIO_TAMANO + _ + 16, 4):** Lee el tamaño del archivo desde el cluster actual más el desplazamiento '_' más 16 con longitud 4 bytes

- **LIST_DIRECTORIO:** Con esta función en listará el contenido del directorio, además ignorará las entradas vacías.

```
def LIST_DIRECTORIO(fiunamfs_img):
    with open(fiunamfs_img, 'rb') as f:
        for _ in range(0, DIRECTORIO_TAMANO, TAM_ENTRADA):
            nombre = CADENA(DIRECTORIO_TAMANO + _,15)
            tam= ENTERO(DIRECTORIO_TAMANO+_+16,4)
            if nombre != '/'*15:
                print("\033[1m   Nombre\t\tTamaño   \033[0m")
                print(f"   {nombre}\t{tam} bytes")
```

- **COPY_TO_FIUNAM:** Lo que realizará esta función es la copia de archivos desde fiunamfs.img al directorio local de nuestro sistema, contará con uso de semáforo, su representación en ASCII, en caso de no encontrar el archivo, se le avisará al usuario por medio de un mensaje.

```
#Copiar a FIUNAM un archivo desde nuestro sistema
def COPY_TO_FIUNAM(fiunamfs_img, archivo_origen, nombre_destino):
    semaforo.acquire()
    with open(fiunamfs_img, 'r+b') as f:
        tam_origen = os.path.getsize(archivo_origen)
        cluster_libre = 5
        posicion_entrada_libre = None

        f.seek(DIRECTORIO_INICIO)
        for _ in range(DIRECTORIO_TAMANO // TAM_ENTRADA):
            posicion_actual = f.tell()
            entrada = f.read(TAM_ENTRADA)
            tipo_archivo = entrada[0:1]
            cluster_ini = struct.unpack('<I', entrada[20:24])[0]

            #Entrada vacia
            if tipo_archivo == b'/' and posicion_entrada_libre is None:
                posicion_entrada_libre = posicion_actual
                #Depurar
                print(f"Encontrada entrada libre en posición {posicion_entrada_libre}")

            if cluster_ini >= cluster_libre:
                cluster_libre = cluster_ini + 1
                print(f"Clouster libre: {cluster_libre}")

        if posicion_entrada_libre is None:
            raise Exception("No hay espacio suficiente")
        else:
            print(f"Entrada libre en: {posicion_entrada_libre}, Cluster libre para el archivo: {cluster_libre}")
```

```
        with look:
            with open(archivo_origen, 'rb') as archivo_origen_f:
                f.seek(cluster_libre * TAMANO_CLUSTER)
                f.write(archivo_origen_f.read())
        f.seek(posicion_entrada_libre)
        f.write(b'-' + nombre_destino.ljust(15).encode('ascii'))
        f.write(struct.pack('<I', tam_origen))
        f.write(struct.pack('<I', cluster_libre))
    semaforo.release()
```

- **COPY_TO_SYSTEM:** Como su nombre lo indica, lo que realizará esta función es copiar un archivo desde el directorio local a FiUnamFS a nuestro sistema, de igual manera, se hará uso de un semáforo y se verificará si el archivo cabe en el sistema de archivos de no ser así lanza un mensaje.

```
def COPY_TO_SYSTEM(fiunamfs_img, nombre_archivo, destino):

    #Adquirir semaforo
    semaforo.acquire()
    with open(fiunamfs_img, 'rb') as f:
        f.seek(DIRECTORIO_INICIO)
        for _ in range(0, DIRECTORIO_TAMANO * 4, TAM_ENTRADA):
            entrada = f.read(TAM_ENTRADA)
            tipo_archivo = entrada[0:1]
            if tipo_archivo == b'-':
                nombre, tam, cluster_ini = (
                    entrada[1:16].decode('ascii').rstrip(),
                    struct.unpack('<I', entrada[16:20])[0],
                    struct.unpack('<I', entrada[20:24])[0]
                )
                #Depurar
                print(f"Nombre encontrado: {nombre}, Tamaño: {tam}, Cluster inicial: {cluster_ini}")
                if nombre.rstrip('\x00').strip() == nombre_archivo.rstrip('\x00').strip():
                    f.seek(cluster_ini * TAMANO_CLUSTER)
                    datos = f.read(tam)
                    with open(destino, 'wb') as archivo_destino:
                        archivo_destino.write(datos)
                    return
    semaforo.release()
    raise FileNotFoundError("El archivo no se encuentra FiUnamFS")
```

- **DELETE:** Esta función eliminará archivos de FiUnamFS, usará un semáforo y además funcionará con representación ASCII e iterará sobre las entradas del directorio en bloques de TAM_ENTRADA.

```
def DELETE(fiunamfs_img, nombre_archivo):
    semaforo.acquire()
    with open(fiunamfs_img, 'r+b') as f:
        f.seek(DIRECTORIO_INICIO)
        for _ in range(DIRECTORIO_TAMANO // TAM_ENTRADA):
            posicion = f.tell()
            entrada = f.read(TAM_ENTRADA)
            nombre = entrada[1:16].decode('ascii').rstrip()
            if nombre.rstrip('\x00').strip() == nombre_archivo.rstrip('\x00').strip():
                f.seek(posicion)
                f.write(b'/' + b' ' * 15)
                print("Archivo eliminado con exito")
                return
    semaforo.release()
    #Exepcion
    raise FileNotFoundError("Archivo no encontrado en FiUnamFS")
```

- **menu_principal:** Interfaz de usuario que muestra el menú para que el usuario pueda escoger alguna de las opciones a la hora de compilar el programa, además de que en cada una de las opciones se hace el llamado de las demás funciones que ya se han declarado.

```
#Menu principar/Interfaz de usuario
def menu_principal():
    while True:
        print("\n- Sistema de Archivos FiUnamFS -")
        print("1. Listar directorio")
        print("2. Copiar archivo de FiUnamFS al sistema")
        print("3. Copiar archivo del sistema a FiUnamFS")
        print("4. Eliminar archivo de FiUnamFS")
        print("5. Salir")
        opcion = input("Selecciona una opción: ")

        if opcion == '1':
            LIST_DIRECTORIO(fiunamfs_img)

        elif opcion == '2':
            nombre_archivo = input("Ingresa el nombre del archivo en FiUnamFS(incluye extencion): ")
            destino = input("Ingresa la ruta de destino en el sistema (ENTER para la carpeta actual): ")
            if not destino:
                destino = os.path.join(os.getcwd(), nombre_archivo)
            try:
                COPY_TO_SYSTEM(fiunamfs_img, nombre_archivo, destino)
                print("Archivo copiado con éxito a", destino)
            #Manejo de exepciones
            except Exception as e:
                print("Error al copiar el archivo:", e)

        elif opcion == '3':
            origen = input("Ingresa la ruta del archivo en el sistema: ")
            nombre_destino = input("Ingresa el nombre del archivo en FiUnamFS: ")
            COPY_TO_FIUNAM(fiunamfs_img, origen, nombre_destino)
        elif opcion == '4':
            nombre_archivo = input("Ingresa el nombre del archivo a eliminar de FiUnamFS(Incluye extension): ")
            DELETE(fiunamfs_img, nombre_archivo)
        elif opcion == '5':
            print("Vuelve pronto!")
            break
        else:
            print("Opción no válida. Vueleve aintentarlo")
```

- Por último se tiene la ejecución del programa en donde se llama a Menu

```
# Archivo actualmente usando
fiunamfs_img = "fiunamfs.img"
# Regresar el menu para ejecutar
menu_principal()
```

Funcionamiento:

En primera instancia se muestra el contenido del directorio, además de que se nos muestra el nombre del sistema de archivos, el tamaño de un cluster, el número de clusters que mide el directorio

```
- Sistema de Archivos FiUnamFS -
1. Listar directorio
2. Copiar archivo de FiUnamFS al sistema
3. Copiar archivo del sistema a FiUnamFS
4. Eliminar archivo de FiUnamFS
5. Salir

Selecciona una opción:
```

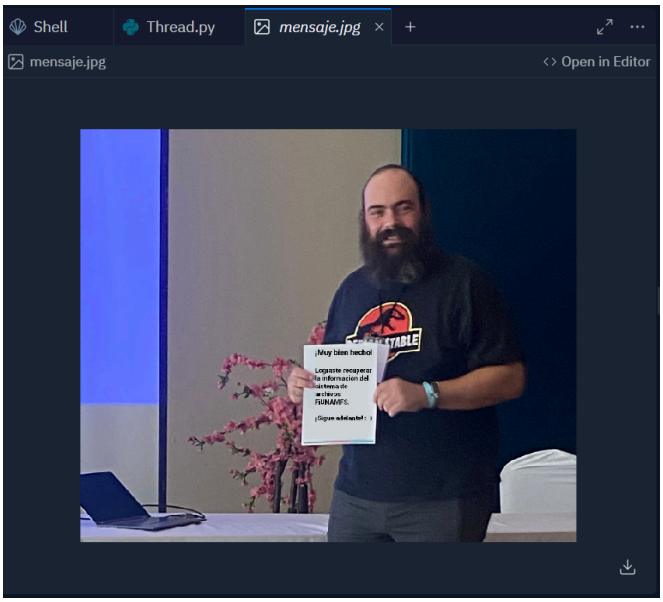
En caso de elegir la opción 1, se muestra el contenido del directorio:

```
- Sistema de Archivos FiUnamFS -
1. Listar directorio
2. Copiar archivo de FiUnamFS al sistema
3. Copiar archivo del sistema a FiUnamFS
4. Eliminar archivo de FiUnamFS
5. Salir

Selecciona una opción: 1
   Nombre       Tamaño
   -README.org  31222 bytes
   Nombre       Tamaño
   /##############  0 bytes
   Nombre       Tamaño
   -logo.png    126423 bytes
   Nombre       Tamaño
   /##############  0 bytes
   Nombre       Tamaño
   /##############  0 bytes
   Nombre       Tamaño
   -mensaje.jpg 254484 bytes
```

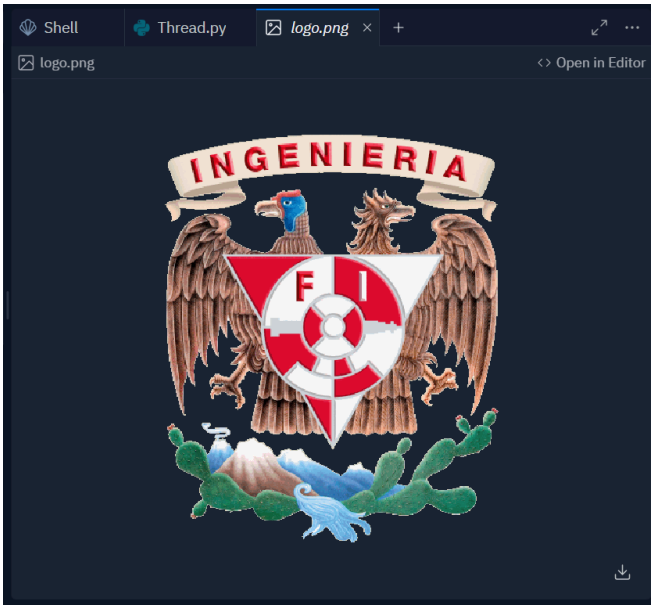
En caso de ingresar la opción 2, se copiara un archivo del sistema a la computadora

```
Selecciona una opción: 2

Ingresa el nombre del archivo en FiUnamFS(incluye
extencion): logo.png

Ingresa la ruta de destino en el sistema (ENTER
para la carpeta actual):
Nombre encontrado: README.org     , Tamaño: 31222,
Cluster inicial: 6
Nombre encontrado: logo.png       , Tamaño: 126423,
Cluster inicial: 22
Archivo copiado con éxito a C:\Users\User\Documents
\Sistemas Operativos 2024-2\sistop-2024-2\proyectos
\1\FloresEvelyn-VeraMarisol\logo.png
```

Las imágenes que se muestran son las siguientes:



(mensaje.jpg)



(logo.png)

De igual manera, se pueden agregar y eliminar archivos del sistema.
Y finalmente la opción 5, nos ayuda a salir y terminar la ejecución.

Conclusiones:

Vera Garmendia Miriam Marisol

En la elaboración del proyecto aprendí acerca de la concurrencia entre hilos, principalmente manejada en python, además de ello, el manejo de la memoria e interacción con el sistema operativo de nuestros equipos de trabajo mediante la importación de las librerías necesarias, siendo este el caso OS. Al manejar el micro sistemas de archivos realiza el código necesario para poder manipularlos en este caso hasta modificarlos y poder descifrar que archivos se contienen en ellos.

Flores Melquiades Evelyn jasmin

Este proyecto de de ayuda para la comprensión de los micro sistemas de archivos debido a que se fue posible el manejarlos y comprender cuál eran sus componentes y principales elementos para así poder saber que contenían en ellos. Además de ello, conocí la manera de ingresar al sistema operativo de nuestras computadoras para que ésta interactúe con los sistemas que se nos proporcionen, para esta ocasión el micro sistemas del profesor. Por último pude observar cómo funcionaba la concurrencia entre hilos con el uso de los semáforos y look, y como estos son de ayuda al momento de asignación e interacción con la memoria del sistema.