Guerrero Gutiérrez Héctor(319303256) **PROYECTO 2** Sistemas operativos G6
López Ramírez Joel Angel(319155103)

Proyecto: (Micro) sistema de archivos multihilos

Documentación del Código para el Sistema de Archivos FiUnamFS

Este documento describe el código de implementación del sistema de archivos FiUnamFS usando Python y la biblioteca FUSE para la creación de un sistema de archivos virtual. A continuación, se detallan las funciones y clases principales, incluyendo el propósito de cada una y su funcionamiento.

Descripción del proyecto

Trabajaremos este proyecto conjunto para las últimas dos unidades:

Sistemas de archivos

Resulta natural que el proyecto sea implementar un sistema de archivos 😉 Para esto, lo harán trabajando sobre una especificación y sobre un caso de referencia.

Administración de procesos

Para esta unidad toca dividir la lógica de un proceso en sus componentes concurrentes, buscando que se comuniquen los cambios de estado empleando mecanismos de sincronización.

Librerías Importadas

El código utiliza varias bibliotecas, tales como:

- logging: Para registrar eventos y depuración del sistema.
- os: Para operaciones de sistema de archivos como obtener ID de usuario y grupo.
- sys: Para manejar argumentos de línea de comandos.
- errno: Para manejar errores específicos del sistema operativo.
- stat: Para definir permisos y modos de archivo.
- math: Para cálculos matemáticos.
- time y datetime: Para manejar fechas y tiempos, utilizados en metadatos de archivos.
- fuse: Biblioteca FUSE para interactuar con el sistema operativo y permitir que el sistema de archivos sea montado.

Funciones Auxiliares

- btoi(b): Convierte un array de 32 bits en formato little-endian a un entero, necesario para leer valores de bytes del sistema de archivos.
- itob(i): Convierte un número entero en un array de 32 bits en formato little-endian, necesario para escribir valores en el sistema de archivos.

Clases de Excepciones Personalizadas

Se definen excepciones específicas para manejar errores comunes en FiUnamFS:

- NotFiUnamPartitionExc: Se lanza cuando el archivo de imagen no tiene la firma de FiUnamFS.
- UnsupportedVersionExc: Para manejar versiones no compatibles del sistema de archivos.
- TruncatedImageExc: Para imágenes con un tamaño menor al mínimo requerido.
- NameTooLargeExc: Se lanza si el nombre de un archivo excede los 14 caracteres permitidos.

Clase FiUnamArchivo

La clase FiUnamArchivo representa una entrada de archivo en el directorio del sistema de archivos.

Atributos:

- nombre: Nombre del archivo, limitado a 14 caracteres.
- tamano: Tamaño del archivo en bytes.
- cluster_ini: Cluster inicial donde el archivo comienza.
- fecha_creacion y fecha_modificacion: Fechas de creación y modificación del archivo.

Métodos:

- __init__(self, b): Inicializa un archivo usando datos binarios o una tupla con nombre y cluster inicial.
- tobytes(self): Convierte los atributos del archivo en una estructura de bytes, lista para ser escrita en disco.

Clase FiUnamFS

La clase FiUnamFS hereda de LoggingMixIn y Operations para implementar el sistema de archivos en FUSE.

Atributos:

- etiqueta: Etiqueta del sistema de archivos.
- cluster: Tamaño de cada cluster en bytes.
- tam_directorio: Tamaño total del directorio en clusters.
- tam_fs: Tamaño total del sistema de archivos.
- entradas: Diccionario que contiene las entradas de archivos.
- entradas_vacias y clusters_ocupados: Conjuntos para gestionar clusters libres y ocupados.

Métodos de la Clase FiUnamFS

- __init__(self, f: str): Inicializa el sistema de archivos leyendo los metadatos desde un archivo de imagen.

Comprueba si el archivo es una imagen de FiUnamFS.

Verifica la versión y configura los atributos principales, como el tamaño de los clusters y la estructura del directorio.

- _existe(self, f: str): Comprueba si un archivo existe en el directorio y devuelve el índice de su entrada.
- _reservar(self, tam: int): Encuentra un espacio libre del tamaño adecuado para almacenar un archivo de tamaño tam. Devuelve el índice del primer cluster disponible.
- access(self, path, mode): Verifica si un archivo especificado en path es accesible.
- getattr(self, path, fh=None): Obtiene los atributos de un archivo o directorio, tales como permisos, tamaño, fechas de creación y modificación. Devuelve un diccionario con los atributos.
- readdir(self, path, fh): Lee el contenido de un directorio, devolviendo los nombres de los archivos contenidos en él.
- read(self, path, length, offset, fh): Lee length bytes desde el archivo especificado en path, comenzando desde offset. Devuelve los datos leídos.
- write(self, path, buf, offset, fh): Escribe datos en un archivo en la ubicación especificada en offset. Actualiza el tamaño y la estructura del directorio.
- truncate(self, path, length, fh=None): Ajusta el tamaño de un archivo, ya sea recortando o expandiendo su tamaño.
- flush(self, path, fh) y fsync(self, path, fdatasync, fh): Métodos implementados para FUSE, pero no tienen efecto en este sistema de archivos ya que los cambios se realizan de inmediato.

Fragmentos de Código y Funcionalidad

1. ***Conversión de bytes a enteros y viceversa***

```python
def btoi(b):
    """
    Convierte un array de 32 bits little-endian a un tipo int
    """
    return b[0]+(b[1]256)+(b[2](256*2))+(b[3](256**3))

def itob(i):
    """
    Convierte un tipo int a un array de 32 bits little-endian
    """
    ba = bytearray()
    op = i
    p3 = op//(256**3)
    op -= p3*(256**3)
    p2 = op//(256**2)
    op -= p2*(256**2)
    p1 = op//256
    op -= p1*256
    ba.extend((op, p1, p2, p3))
    return ba
```

Función: Estas funciones convierten valores entre un entero y su representación en bytes (32 bits en formato little-endian).

Utilidad: Son esenciales para leer y escribir valores numéricos en la imagen del sistema de archivos en el formato correcto.

2. *Clase FiUnamArchivo*

```python
class FiUnamArchivo:
    """
    Entrada en el directorio
    """
    def _init_(self, b):
        if type(b) == bytes:
            self.nombre = b[1:15].decode(encoding="us-ascii").strip()
            self.tamano = btoi(b[16:19]+bytes(1)) # NOTE: 4 o 3 bytes
            self.cluster_ini = btoi(b[20:23]+bytes(1)) # Idem
            self.fecha_creacion = datetime.strptime(b[24:38].decode(), "%Y%m%d%H%M%S")
            self.fecha_modificacion = datetime.strptime(b[38:52].decode(), "%Y%m%d%H%M%S")
        elif type(b) == tuple:
            self.nombre = b[0]
            self.tamano = 0
            self.cluster_ini = b[1]
            self.fecha_creacion = datetime.now()
            self.fecha_modificacion = datetime.now()
```

Descripción: La clase modela un archivo en el sistema de archivos FiUnamFS.

Atributos:

- nombre: Nombre del archivo.
- tamano: Tamaño en bytes.
- cluster_ini: Cluster inicial.
- fecha_creacion y fecha_modificacion: Fechas de creación y modificación.

Funcionalidad: Se utiliza para almacenar y manipular los metadatos de archivos en el sistema de archivos.

3. *Inicialización del sistema de archivos*

```python
def _init_(self, f: str):
    if os.path.getsize(f) < 54:
        raise TruncatedImageExc()

    self.imagen = open(f, 'rb+')

    # Firma
    if not self.imagen.read(8) == b"FiUnamFS":
        raise NotFiUnamPartitionExc()

    # Version
    self.imagen.seek(10)
    if not self.imagen.read(4) == b"25-1":
        raise UnsupportedVersionExc()

    # Etiqueta
    self.imagen.seek(20)
    self.etiqueta = self.imagen.read(15)

    # Tamaño de cluster
    self.imagen.seek(40)
    self.cluster = btoi(self.imagen.read(4))

    # Tamaño de directorio
    self.imagen.seek(45)
    self.tam_directorio = btoi(self.imagen.read(4))
    #continua.......... (para brevedad de documentacion)
```

Función: Este método inicializa el sistema de archivos verificando la firma y versión de FiUnamFS.

Validaciones: Comprueba si la imagen contiene una firma válida, la versión correcta y el tamaño mínimo.

Excepciones: Lanza errores personalizados (TruncatedImageExc, NotFiUnamPartitionExc, UnsupportedVersionExc) si la imagen es inválida.

4. Función getattr

```python
def getattr(self, path, fh=None):
    print(path)

    inodo = self._existe(path)
    if path == "/":
        ahora = datetime.now()
        return dict(
            st_mode=(stat.S_IRWXU|stat.S_IRWXG|stat.S_IRWXO|stat.S_IFDIR), # All permissions as in ntfs
            st_ctime=time.mktime(ahora.timetuple()),
            st_mtime=time.mktime(ahora.timetuple()),
            st_atime=time.mktime(ahora.timetuple()),
            st_nlink=2,
            st_gid=os.getgid(),
            st_uid=os.getuid()
        )

    elif path.startswith("/.") or path.startswith("/autorun.inf"):
        return dict(

            st_gid=os.getgid(),
            st_ino=2000,
            st_mode=(stat.S_IRWXU|stat.S_IRWXG|stat.S_IRWXO|stat.S_IFREG),
            st_nlink=1,
            st_size=0,
            st_uid=os.getuid()
        )
    elif inodo is None:
        raise FuseOSError(errno.ENOENT)
    else:
        print("reconoce")
        return dict(
            st_atime=time.mktime(self.entradas[inodo].fecha_modificacion.timetuple()),
            st_ctime=time.mktime(self.entradas[inodo].fecha_creacion.timetuple()),
            st_gid=os.getgid(),
            st_ino=inodo,
            st_mode=(stat.S_IRWXU|stat.S_IRWXG|stat.S_IRWXO|stat.S_IFREG),
            st_mtime=time.mktime(self.entradas[inodo].fecha_modificacion.timetuple()),
            st_nlink=1,
            st_size=self.entradas[inodo].tamano,
            st_uid=os.getuid()
        )
```

Función: Obtiene los atributos de un archivo o directorio en el sistema de archivos.

Retorno: Devuelve un diccionario con los atributos del archivo, como permisos, tamaño y fechas.

Uso: Es una función importante en FUSE para acceder a metadatos.

5. ***Función readdir***

```
def readdir(self, path, fh):
    lista = [ ".", ".." ]
    for e in self.entradas.values():
        lista.append(e.nombre)
        print("Archivos listados en readdir:", lista)
    return lista
```

Función: Lee el contenido de un directorio y devuelve una lista con los nombres de los archivos.

Uso: Permite listar los archivos en un directorio cuando se monta el sistema de archivos.

6. ***Función read***

```
def read(self, path, length, offset, fh):
    inodo = self._existe(path)
    self.imagen.seek(self.entradas[inodo].cluster_ini * self.cluster + (offset or 0))
    return self.imagen.read(length or self.entradas[inodo].tamano)
```

Función: Lee un número específico de bytes (length) desde una posición (offset) en el archivo.

Uso: Utilizada para obtener el contenido de un archivo al acceder a él en el sistema de archivos montado.

7. ***Función write***

```
def write(self, path, buf, offset, fh):
    inodo = self._existe(path)
    # Escribir en archivo
    self.imagen.seek(self.entradas[inodo].cluster_ini * self.cluster + (offset or 0))
    self.imagen.write(buf)
    self.entradas[inodo].tamano = len(buf) + (offset or 0)
    # Escribir en directorio
    self.imagen.seek(self.cluster+64*inodo)
    self.imagen.write(self.entradas[inodo].tobytes())
    return len(buf)
```

Función: Escribe datos (buf) en el archivo en una ubicación (offset) y actualiza su tamaño.

Uso: Permite modificar el contenido de archivos en el sistema de archivos.

8. *Ejecución de hilos concurrentes*

Se crean dos hilos, uno llamado 'hilo_estado' e 'hilo_supervision' los cuales los vamos a iniciar en el constructor. Para la ejecución creamos dos métodos:

Método 'monitor_estado':

```python
def monitor_estado(self):
    """
    Hilo que monitorea el estado del sistema de archivos
    """
    while True:
        with self.lock:
            print("Estado actual de entradas:", list(self.entradas.keys()))
        time.sleep(5)
```

Imprime el estado actual de las entradas (el número de archivos y su identificador) en donde se refresca cada 5 segundos. Utilizamos loch para asegurar que el acceso a entradas sea seguro y que no tengamos conflictos con los hilos.

Método 'supervisar_directorio':

```python
def supervisar_directorio(self):
    """
    Hilo que supervisa cambios en el directorio
    """
    while True:
        with self.lock:
            print("Clusters libres:", self.entradas_vacias)
        time.sleep(10)
```

Imprime los clusters libres cada 10 segundos, se usa lock para asegurar el acceso concurrente.

Funcionamiento:

```
, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63}
Estado actual de entradas: [1, 2, 3, 4]
Estado actual de entradas: [1, 2, 3, 4]
DEBUG:fuse.log-mixin:-> getattr / (None,)
3
/
{1: <__main__.FiUnamArchivo object at 0x7965006d3460>, 2: <__main__.FiUnamArchivo object at 0x7965006d2ef0>, 3: <__main__.FiUnamArchivo object at 0x7965004c1990>, 4: <__main__.FiUnamArchivo object at 0x79
65004c1a80>}
DEBUG:fuse.log-mixin:<- getattr {'st_mode': 16895, 'st_ctime': 1731033122.0, 'st_mtime': 1731033122.0, 'st_atime': 1731033122.0, 'st_nlink': 2, 'st_gid': 1000, 'st_uid': 1000}
Clusters libres: {6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53
, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63}
Estado actual de entradas: [1, 2, 3, 4]
Estado actual de entradas: [1, 2, 3, 4]
Clusters libres: {6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53
, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63}
Estado actual de entradas: [1, 2, 3, 4]
Estado actual de entradas: [1, 2, 3, 4]
Clusters libres: {6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53
, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63}
Estado actual de entradas: [1, 2, 3, 4]
Estado actual de entradas: [1, 2, 3, 4]
DEBUG:fuse.log-mixin:-> getattr / (None,)
3
/
{1: <__main__.FiUnamArchivo object at 0x7965006d3460>, 2: <__main__.FiUnamArchivo object at 0x7965006d2ef0>, 3: <__main__.FiUnamArchivo object at 0x7965004c1990>, 4: <__main__.FiUnamArchivo object at 0x79
65004c1a80>}
DEBUG:fuse.log-mixin:<- getattr {'st_mode': 16895, 'st_ctime': 1731033148.0, 'st_mtime': 1731033148.0, 'st_atime': 1731033148.0, 'st_nlink': 2, 'st_gid': 1000, 'st_uid': 1000}
Clusters libres: {6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53
, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63}
Estado actual de entradas: [1, 2, 3, 4]
Estado actual de entradas: [1, 2, 3, 4]
Clusters libres: {6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53
, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63}
Estado actual de entradas: [1, 2, 3, 4]
Estado actual de entradas: [1, 2, 3, 4]
DEBUG:fuse.log-mixin:-> getattr / (None,)
3
/
{1: <__main__.FiUnamArchivo object at 0x7965006d3460>, 2: <__main__.FiUnamArchivo object at 0x7965006d2ef0>, 3: <__main__.FiUnamArchivo object at 0x7965004c1990>, 4: <__main__.FiUnamArchivo object at 0x79
65004c1a80>}
DEBUG:fuse.log-mixin:<- getattr {'st_mode': 16895, 'st_ctime': 1731033171.0, 'st_mtime': 1731033171.0, 'st_atime': 1731033171.0, 'st_nlink': 2, 'st_gid': 1000, 'st_uid': 1000}
DEBUG:fuse.log-mixin:-> getattr / (None,)
3
/
{1: <__main__.FiUnamArchivo object at 0x7965006d3460>, 2: <__main__.FiUnamArchivo object at 0x7965006d2ef0>, 3: <__main__.FiUnamArchivo object at 0x7965004c1990>, 4: <__main__.FiUnamArchivo object at 0x79
65004c1a80>}
DEBUG:fuse.log-mixin:<- getattr {'st_mode': 16895, 'st_ctime': 1731033171.0, 'st_mtime': 1731033171.0, 'st_atime': 1731033171.0, 'st_nlink': 2, 'st_gid': 1000, 'st_uid': 1000}
Clusters libres: {6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53
, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63}
Estado actual de entradas: [1, 2, 3, 4]
Estado actual de entradas: [1, 2, 3, 4]
```

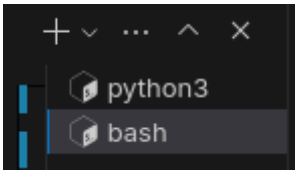
En la ejecución se observa la salida del número de entradas y su identificador, en el mismo si borramos algún archivo el número de entradas va a disminuir. Al igual que se imprimen los clusters disponibles.

Indicaciones de ejecución.

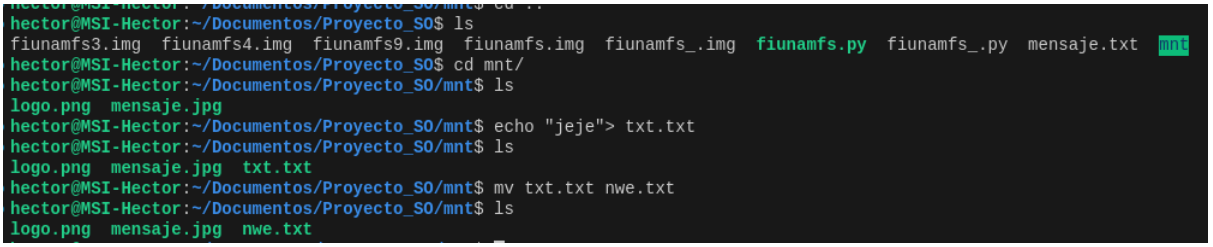
python3 fiunamfs_.py 'nombre_imagen' 'punto_montaje'

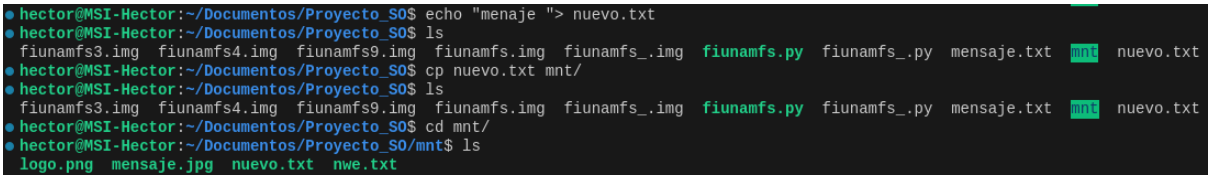
Ejemplo de ejecución del comando:

Posteriormente se recomienda abrir otra ventana de ejecución paralela para poder ejecutar las funciones del ejecutable.



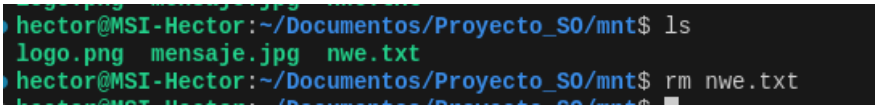
Ejempls de una función de la ejecución:



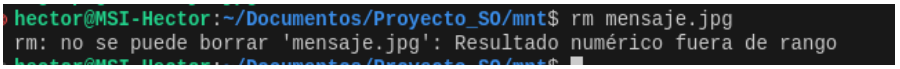


NOTAS:

La función rm funciona de manera correcta si la realizamos manualmente directo sobre la imagen, sin embargo es inestable, luego funciona y en ocasiones no. Adjuntamos una muestra de su funcionamiento.



En ocasiones surge la siguiente advertencia:



Sin más las especificaciones pedidas son posibles hacerlas, se adjunta evidencia de la apertura de la imágen.

Referencias:

- Crysol. (2011, abril 7). *FUSE y Python: crea tu propio sistema de ficheros fácilmente*. Crysol.org. https://crysol.org/recipe/2011-04-07/fuse-y-python-crea-tu-propio-sistema-de-ficheros-fcilmente.html
- GeeksforGeeks. (s. f.). *Python | os.stat() method*. GeeksforGeeks. Recuperado el 7 de noviembre de 2024, de https://www.geeksforgeeks.org/python-os-stat-method/
- Python Software Foundation. (s. f.). *stat — Interpretación de resultados de stat()*. Documentación de Python 3. Recuperado el 7 de noviembre de 2024, de https://docs.python.org/es/3/library/stat.html