LABORATORIO MANEJO E IMPLEMENTACION DE ARCHIVOS - Sección B

Marcelo Andre Juarez Alfaro - 202010367

Manual Técnico - Proyecto 2: Sistema de Archivos EXT2/EXT3

Descripción de la Arquitectura del Sistema

El sistema implementa una arquitectura cliente-servidor para simular un sistema de archivos EXT2 y EXT3, con un backend desarrollado en Go usando Fiber y un frontend en Next.js. En esta Fase 2, se ha añadido soporte para EXT3 con Journaling, nuevos comandos, y un visualizador gráfico en el frontend para navegar por el sistema de archivos. Además, la aplicación se ha desplegado en AWS: el frontend en un bucket S3 y el backend en una instancia EC2 dockerizada. La comunicación entre el frontend y el backend se realiza mediante una API RESTful sobre HTTP.

Componentes Principales

1. **Frontend (Next.js)**
    - **Tecnología**: Next.js con React, renderizado del lado del cliente.
    - **Estructura**:
        - page.tsx: Página principal que integra la interfaz con componentes como InputTerminal.tsx (entrada de comandos), OutputTerminal.tsx (salida de resultados) y FileUpload.tsx (carga de scripts). También incluye nuevas vistas para iniciar sesión y visualizar el sistema de archivos.
        - api.ts: Servicio que realiza solicitudes POST a http://<ip-ec2>:3000/execute.
    - **Funcionalidad**:
        - Permite al usuario ingresar comandos o cargar scripts y muestra los resultados devueltos por el backend.
        - Nueva interfaz gráfica para login (reemplaza el comando LOGIN).
        - Visualizador del sistema de archivos en modo lectura, permitiendo navegar por discos, particiones, carpetas y archivos.
    - **Despliegue**: Hospedado en un bucket S3 de AWS, accesible mediante una URL pública.
2. **Backend (Go + Fiber)**
    - **Tecnología**: Go con Fiber, un framework ligero para APIs RESTful.
    - **Estructura**:
        - main.go: Configura el servidor en el puerto 3000 con un endpoint /execute (POST). Parsea comandos y los pasa al analyzer.
        - analyzer: Interpreta y ejecuta comandos (por ejemplo, MKDISK, FDISK), coordinando las operaciones sobre las estructuras.
        - structures: Define las estructuras EXT2/EXT3 (MBR, SuperBlock, EBR, Journal, etc.) con métodos de serialización/deserialización.
        - stores: Gestiona el estado en memoria (MountedPartitions, CurrentSession).
        - utils: Ofrece funciones auxiliares (por ejemplo, ConvertToBytes).
        - commands: Contiene la lógica de los comandos (por ejemplo, mkfs.go, chgrp.go).
    - **Funcionalidad**:
        - Procesa comandos enviados desde el frontend.

        - Gestiona un sistema de archivos EXT2/EXT3 con Journaling.
        - Almacena datos en archivos binarios .mia.
        - Devuelve resultados en formato JSON.
    - **Despliegue**: Dockerizado y desplegado en una instancia EC2 de AWS con Ubuntu 22.04 LTS, escuchando en el puerto 3000.
3. **Sistema EXT2/EXT3 en Disco**
    - **Estructura**: Archivos .mia que almacenan las estructuras persistentes (MBR, Partition, EBR, SuperBlock, Bitmap, Inode, FolderBlock, FileBlock, PointerBlock, Journal).
    - **Funcionalidad**: Representa el sistema de archivos EXT2/EXT3 en almacenamiento físico, gestionado por el backend.

Interacción entre Componentes

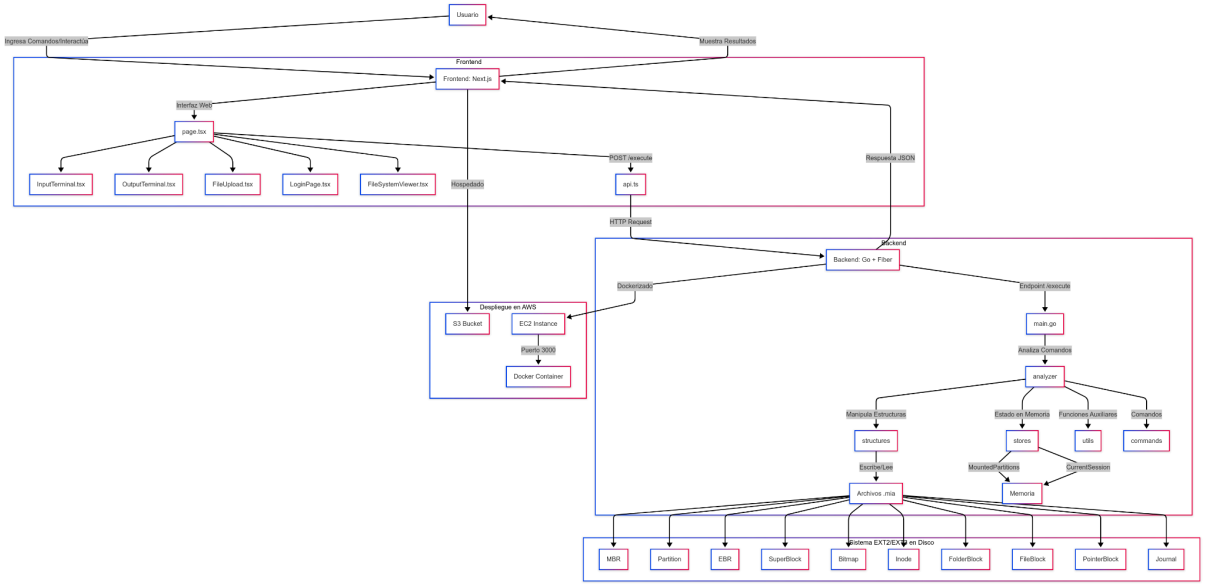
- **Flujo**:
    1. El usuario interactúa con el frontend a través de InputTerminal.tsx, FileUpload.tsx, o la interfaz gráfica de login/visualización.
    2. page.tsx envía los comandos al backend vía api.ts (POST /execute).
    3. El backend (main.go) procesa los comandos con analyzer, manipulando las estructuras en structures y actualizando el estado en stores.
    4. Las operaciones se reflejan en los archivos .mia, que contienen el sistema EXT2/EXT3.
    5. El backend devuelve un JSON con los resultados, que el frontend muestra en OutputTerminal.tsx o en el visualizador gráfico.
- **Comunicación**: API RESTful (HTTP), con solicitudes como {"command": "mkdisk -size=10 -unit=M"} y respuestas como {"output": "Disco creado"}.

Despliegue en AWS

- **Frontend**:
    - Hospedado en un bucket S3 de AWS, configurado para acceso público como un sitio web estático.
    - URL de acceso: Proporcionada por S3 (por ejemplo, http://<bucket-name>.s3-website-us-east-1.amazonaws.com).
- **Backend**:
    - Desplegado en una instancia EC2 (t2.micro, Ubuntu 22.04 LTS).
    - Dockerizado usando una imagen (go-fiber-app) creada a partir del código del backend.
    - El contenedor corre en el puerto 3000, accesible mediante la IP pública del EC2 (por ejemplo, http://<ip-ec2>:3000).
    - Security Group configurado para permitir tráfico en los puertos 22 (SSH) y 3000 (backend).

Diagrama de Arquitectura

El siguiente diagrama ilustra el flujo de datos y las relaciones entre componentes, incluyendo el despliegue en AWS:



Explicación del Diagrama:

- **Usuario → Frontend**: Ingreso de comandos y visualización de resultados a través de la interfaz web.
- **Frontend → Backend**: Solicitudes HTTP desde api.ts al endpoint /execute.
- **Backend → Sistema EXT2/EXT3**: Manipulación de estructuras en archivos .mia mediante structures, con soporte de stores, utils, y commands.
- **Backend → Frontend**: Respuesta JSON que regresa al usuario.
- **Despliegue en AWS**: El frontend se aloja en S3, mientras que el backend se ejecuta en un contenedor Docker en EC2.

Explicación de las Estructuras de Datos

A continuación, se describen las estructuras de datos fundamentales utilizadas para simular el sistema de archivos EXT2/EXT3 en el proyecto. Estas estructuras se almacenan en archivos binarios con extensión .mia, que representan discos virtuales. También se incluyen utilidades y estructuras auxiliares definidas en los paquetes utils y stores.

Master Boot Record (MBR)

El MBR es la estructura inicial de cada disco virtual, escrita en el primer sector del archivo .mia.

- **Función**: Contiene metadatos del disco y define hasta cuatro particiones primarias o extendidas.
- **Campos**:
  - mbr_tamano (int32): Tamaño total del disco en bytes.
  - mbr_fecha_creacion (float32): Fecha y hora de creación (timestamp).
  - mbr_disk_signature (int32): Identificador único (aleatorio).

    - dsk_fit ([1]byte): Tipo de ajuste del disco ('B', 'F', 'W').
    - mbr_partitions ([4]Partition): Arreglo de cuatro estructuras Partition.
- **Organización**: Ocupa un tamaño fijo al inicio del .mia. Las particiones definidas aquí determinan las regiones donde se escriben sistemas EXT2/EXT3 o EBRs.

Partition

La estructura Partition, definida en partition.go, describe una partición primaria o extendida dentro del MBR.

- **Función**: Define las propiedades de una partición (tamaño, tipo, ajuste) y su estado de montaje.
- **Campos**:
    - Part_status ([1]byte): Estado ('N': no usada, '0': creada, '1': montada).
    - Part_type ([1]byte): Tipo ('P': primaria, 'E': extendida).
    - Part_fit ([1]byte): Ajuste ('B', 'F', 'W').
    - Part_start (int32): Byte de inicio de la partición.
    - Part_size (int32): Tamaño en bytes.
    - Part_name ([16]byte): Nombre de la partición.
    - Part_correlative (int32): Correlativo asignado al montar.
    - Part_id ([4]byte): ID asignado al montar (por ejemplo, "671A").
- **Métodos**:
    - CreatePartition(partStart, partSize int32, partType, partFit, partName string): Inicializa una partición con estado '0', asignando Part_start, Part_size, Part_type, Part_fit, y Part_name. Usado por FDISK.
    - MountPartition(correlative int32, id string): Cambia el estado a '1', asigna Part_correlative y Part_id. Invocado por MOUNT.
    - PrintPartition(): Método de depuración que imprime los valores de la partición.
- **Uso**: Creada por FDISK, montada por MOUNT, y su Part_id se almacena en stores.MountedPartitions.

Extended Boot Record (EBR)

El EBR, definido en ebr.go, describe particiones lógicas dentro de una partición extendida, funcionando como una lista enlazada.

- **Función**: Gestiona particiones lógicas, indicando su ubicación, tamaño y enlace al siguiente EBR.
- **Campos**:
    - Part_status ([1]byte): Estado ('N': no usada, '0': creada, '1': montada).
    - Part_fit ([1]byte): Ajuste ('B', 'F', 'W').
    - Part_start (int32): Byte de inicio de la partición lógica.
    - Part_size (int32): Tamaño en bytes.
    - Part_next (int32): Byte del siguiente EBR o -1 si es el último.
    - Part_name ([16]byte): Nombre de la partición lógica.
    - Part_id ([4]byte): ID asignado al montar (por ejemplo, "671B").

- **Organización**: Escrito dentro del espacio de una partición extendida definida en el MBR. Cada EBR ocupa un tamaño fijo y apunta al siguiente mediante Part_next, formando una cadena de particiones lógicas.
- **Métodos**:
    - Serialize(path string, offset int64): Escribe el EBR en el archivo .mia en la posición offset usando binary.Write en formato Little Endian. Usado por FDISK.
    - Deserialize(path string, offset int64): Lee un EBR desde el archivo en la posición offset, calculando su tamaño con binary.Size y usando binary.Read. Usado por MOUNT o REP.
    - Print(): Método de depuración que imprime los valores del EBR.
- **Uso**: Creado por FDISK para particiones lógicas, montado por MOUNT, actualizando Part_status a '1' y asignando Part_id.

SuperBlock

El SuperBlock es la estructura raíz del sistema EXT2/EXT3 en una partición primaria o lógica, definida en superblock.go. Almacena metadatos críticos y controla la creación y actualización de bitmaps.

- **Función**: Proporciona una visión general del sistema EXT2/EXT3, incluyendo el conteo de recursos, su estado y las posiciones de otras estructuras (bitmaps, inodos, bloques, journal).
- **Campos**:
    - S_filesystem_type (int32): Identificador del sistema (2 para EXT2, 3 para EXT3).
    - S_inodes_count (int32): Total de inodos, calculado según el tamaño de la partición.
    - S_blocks_count (int32): Total de bloques (triple de inodos).
    - S_free_inodes_count (int32): Inodos libres.
    - S_free_blocks_count (int32): Bloques libres.
    - S_mtime (float32): Último montaje (timestamp).
    - S_umtime (float32): Último desmontaje (timestamp).
    - S_mnt_count (int32): Veces montado.
    - S_magic (int32): Valor mágico (0xEF53).
    - S_inode_size (int32): Tamaño del inodo (88 bytes).
    - S_block_size (int32): Tamaño del bloque (64 bytes).
    - S_first_ino (int32): Primer inodo libre (inicia en 2 tras crear users.txt).
    - S_first_blo (int32): Primer bloque libre (inicia en 2 tras crear users.txt).
    - S_bm_inode_start (int32): Inicio del bitmap de inodos.
    - S_bm_block_start (int32): Inicio del bitmap de bloques.
    - S_inode_start (int32): Inicio de la tabla de inodos.
    - S_block_start (int32): Inicio de la tabla de bloques.
    - S_journal_start (int32): Inicio del área del Journal (solo EXT3).
    - S_journal_count (int32): Número de entradas actuales en el Journal.
    - S_journal_size (int32): Número máximo de entradas que puede contener el Journal (fijo en 50).
- **Organización**:

    - Escrito al inicio de la partición EXT2/EXT3 (justo después de Partition.Part_start o EBR.Part_start).
    - Seguido por el Journal (en EXT3), los bitmaps, la tabla de inodos y la tabla de bloques, cuyas posiciones se calculan dinámicamente según el tamaño de la partición.
- **Métodos**:
    - CreateBitMaps(file *os.File): Inicializa los bitmaps de inodos y bloques durante el formateo.
    - UpdateBitmapInode(path string, inodeIndex int32): Actualiza el bitmap de inodos, usado al asignar nuevos inodos.
    - UpdateBitmapBlock(path string, blockIndex int32): Actualiza el bitmap de bloques, usado al asignar nuevos bloques.
    - CreateFolder(path string, parentsDir []string, destDir string): Crea carpetas en el sistema de archivos.
- **Uso**: Creado durante MKFS, define la estructura del sistema EXT2/EXT3. Sus campos S_first_ino y S_first_blo se actualizan tras operaciones como CreateUsersFile.

Journal

El Journal, definido en journal.go, es una bitácora que registra todas las operaciones realizadas en el sistema de archivos EXT3.

- **Función**: Permite la recuperación del sistema tras una pérdida, almacenando un historial de operaciones.
- **Campos**:
    - Count (int32): Contador de la entrada (incremental).
    - Content (Information): Información de la operación.
- **Subestructura Information**:
    - Operation ([10]byte): Operación realizada (por ejemplo, "mkdir").
    - Path ([32]byte): Ruta donde se realizó la operación.
    - Content ([64]byte): Contenido (si aplica, por ejemplo, para archivos).
    - Date (float32): Fecha de la operación (timestamp).
- **Organización**:
    - Escrito después del SuperBlock en particiones EXT3.
    - Tamaño fijo de 50 entradas (definido por S_journal_size).
    - Cada entrada ocupa 114 bytes (binary.Size(Journal{})).
- **Métodos**:
    - AddJournalEntry(sb *SuperBlock, diskPath, operation, path, content string): Añade una entrada al Journal, verificando que no exceda S_journal_size.
- **Uso**: Utilizado por comandos como MKFS, MKDIR, MKFILE, etc., para registrar operaciones. Consultable mediante el comando JOURNALING y usado por RECOVERY.

EXT2

El sistema EXT2 no es una estructura única, sino la integración de SuperBlock, Bitmap, Inode, FolderBlock, FileBlock, y PointerBlock, implementada parcialmente en ext2.go. Representa el sistema de archivos completo dentro de una partición.

- **Función**: Simula un sistema de archivos EXT2 con una estructura jerárquica que incluye un directorio raíz (/) y un archivo inicial (users.txt).
- **Organización**:
    - **SuperBlock**: Inicio de la partición.
    - **Bitmap de Inodos**: Seguido del SuperBlock.
    - **Bitmap de Bloques**: Después del bitmap de inodos.
    - **Tabla de Inodos**: Contiene inodos como el raíz (0) y users.txt (1).
    - **Tabla de Bloques**: Aloja bloques de carpetas (raíz) y archivos (users.txt).
- **Método Clave**:
    - CreateUsersFile(path string):
        - Crea el directorio raíz (/) y el archivo users.txt durante el formateo (MKFS).
        - **Raíz (/)**:
            - Inodo 0: Tipo '0' (carpeta), permisos 777, apunta al bloque 0.
            - Bloque 0 (FolderBlock): Entradas . (inodo 0), .. (inodo 0), users.txt (inodo 1).
            - Actualiza Bitmap para inodo 0 y bloque 0.
        - **Archivo** users.txt:
            - Inodo 1: Tipo '1' (archivo), permisos 777, tamaño igual a len("1,G,root\n1,U,root,123\n"), apunta al bloque 1.
            - Bloque 1 (FileBlock): Contiene "1,G,root\n1,U,root,123\n".
            - Actualiza Bitmap para inodo 1 y bloque 1.
        - Actualiza S_first_ino y S_first_blo a 2, indicando los próximos recursos libres.
- **Uso**: Inicializado por MKFS para establecer la base del sistema EXT2, permitiendo operaciones como LOGIN, MKUSR, y MKGRP.

EXT3

El sistema EXT3 extiende EXT2 al incluir soporte para Journaling, lo que permite la recuperación del sistema tras fallos. Es la integración de SuperBlock, Journal, Bitmap, Inode, FolderBlock, FileBlock, y PointerBlock, implementada en ext3.go.

- **Función**: Simula un sistema de archivos EXT3 con una estructura jerárquica que incluye un directorio raíz (/) y un archivo inicial (users.txt), además de registrar operaciones en el Journal para recuperación.
- **Organización**:
    - **SuperBlock**: Inicio de la partición.
    - **Journal**: Seguido del SuperBlock, con un tamaño fijo de 50 entradas.
    - **Bitmap de Inodos**: Después del Journal.
    - **Bitmap de Bloques**: Seguido del bitmap de inodos.
    - **Tabla de Inodos**: Contiene inodos como el raíz (0) y users.txt (1).
    - **Tabla de Bloques**: Aloja bloques de carpetas (raíz) y archivos (users.txt).
- **Cálculo de Estructuras**:
    - El número de inodos (S_inodes_count) y bloques (S_blocks_count) se calcula resolviendo la ecuación:
      tamaño_particion = sizeof(SuperBlock) + S_journal_size * sizeof(Journal) + S_inodes_count + S_blocks_count + S_inodes_count * sizeof(Inode) +

S_blocks_count * sizeof(Block)
Donde S_blocks_count = 3 * S_inodes_count y S_journal_size = 50. El valor final de S_inodes_count se obtiene aplicando la función floor al resultado.

- **Método Clave**:
  - CreateUsersFile(path string):
    - Similar a EXT2, crea el directorio raíz (/) y el archivo users.txt durante el formateo (MKFS).
    - **Raíz (/)**:
      - Inodo 0: Tipo '0' (carpeta), permisos 777, apunta al bloque 0.
      - Bloque 0 (FolderBlock): Entradas . (inodo 0), .. (inodo 0), users.txt (inodo 1).
      - Actualiza Bitmap para inodo 0 y bloque 0.
    - **Archivo** users.txt:
      - Inodo 1: Tipo '1' (archivo), permisos 777, tamaño igual a len("1,G,root\n1,U,root,123\n"), apunta al bloque 1.
      - Bloque 1 (FileBlock): Contiene "1,G,root\n1,U,root,123\n".
      - Actualiza Bitmap para inodo 1 y bloque 1.
    - Actualiza S_first_ino y S_first_blo a 2, indicando los próximos recursos libres.
    - Registra la operación en el Journal con AddJournalEntry.
- **Uso**:
  - Inicializado por MKFS -fs=3fs para establecer la base del sistema EXT3.
  - Permite operaciones como LOGIN, MKUSR, MKGRP, y nuevas funcionalidades como RECOVERY y JOURNALING.
  - El Journal asegura la recuperación del sistema tras fallos simulados con LOSS.

Bitmap

Los Bitmaps son mapas de bits que indican el estado de inodos y bloques dentro del sistema EXT2/EXT3, utilizando '0' para recursos libres y '1' para ocupados. Se implementan como parte de las operaciones del SuperBlock.

- **Función**: Gestionan la asignación de inodos y bloques, permitiendo un seguimiento eficiente de los recursos disponibles.
- **Tipos**:
  - **Bitmap de Inodos**: Un byte por inodo, con un tamaño igual a S_inodes_count. Indica qué inodos están ocupados o libres.
  - **Bitmap de Bloques**: Un byte por bloque, con un tamaño igual a S_blocks_count (triple de inodos). Indica qué bloques están asignados.
- **Organización**:
  - Se escriben después del Journal (en EXT3) o del SuperBlock (en EXT2).
  - Inicio: S_bm_inode_start para inodos y S_bm_block_start para bloques, definidos en el SuperBlock.
- **Métodos**:
  - CreateBitMaps(file *os.File): Inicializa los bitmaps en el archivo .mia durante el formateo (MKFS).

    - UpdateBitmapInode(path string, inodeIndex int32): Actualiza un inodo específico en el bitmap, marcándolo como ocupado ('1').
    - UpdateBitmapBlock(path string, blockIndex int32): Actualiza un bloque específico en el bitmap, marcándolo como ocupado ('1').
- **Uso**: Durante el formateo inicial (MKFS), se crean con CreateBitMaps. Se actualizan dinámicamente al crear archivos o carpetas (por ejemplo, MKFILE, MKDIR).

Inode

El Inode describe un archivo o carpeta en EXT2/EXT3.

- **Función**: Almacena metadatos y apuntadores a bloques.
- **Campos**:
    - I_uid (int32): UID del propietario.
    - I_gid (int32): GID del grupo.
    - I_size (int32): Tamaño en bytes.
    - I_atime (float32): Último acceso (timestamp).
    - I_ctime (float32): Creación (timestamp).
    - I_mtime (float32): Última modificación (timestamp).
    - I_block ([15]int32): Apuntadores (12 directos, 1 simple indirecto, 1 doble indirecto, 1 triple indirecto).
    - I_type ([1]byte): Tipo ('1': archivo, '0': carpeta).
    - I_perm ([3]byte): Permisos UGO (por ejemplo, "664").
- **Organización**: En la tabla de inodos, apunta a bloques gestionados por utils.SplitStringIntoChunks para contenidos.

FolderBlock

El FolderBlock almacena entradas de directorios.

- **Función**: Contiene nombres y apuntadores a inodos.
- **Campos**:
    - B_content ([4]FolderContent):
        - B_name ([12]byte): Nombre.
        - B_inodo (int32): Apuntador al inodo.
- **Organización**: Tamaño de 64 bytes, usado en inodos de carpetas.

FileBlock

El FileBlock guarda contenido de archivos.

- **Función**: Almacena hasta 64 bytes de datos.
- **Campos**:
    - B_content ([64]byte): Contenido.
- **Organización**: Tamaño de 64 bytes, usado en inodos de archivos con ayuda de utils.SplitStringIntoChunks.

PointerBlock

El PointerBlock gestiona apuntadores indirectos.

- **Función**: Contiene 16 apuntadores a otros bloques.
- **Campos**:
    - B_pointers ([16]int32): Apuntadores.
- **Organización**: Tamaño de 64 bytes, usado en niveles indirectos.

Utilidades en utils

El paquete utils proporciona funciones auxiliares para la gestión de estructuras:

- ConvertToBytes(size int, unit string): Convierte tamaños a bytes según la unidad (B, K, M). Usada en MKDISK y FDISK.
- GetLetterAndPartitionCorrelative(path string): Asigna letras y correlativos a particiones para IDs como "671A". Esencial para MOUNT.
- CreateParentDirs(path string): Crea carpetas padre para rutas. Usada en MKDISK, MKFILE, y MKDIR.
- SplitStringIntoChunks(s string): Divide contenido en bloques de 64 bytes para FileBlock.

Estructuras en stores

- MountedPartitions (map[string]string): Almacena particiones montadas (ID → ruta), actualizado por MOUNT y UNMOUNT.
- Session: Estructura para sesiones activas con ID, Username, UID, y GID, gestionada por la interfaz gráfica de login/logout.
- GetMountedPartitionRep(id string): Recupera MBR y SuperBlock para reportes como REP.
- GetMountedPartitionSuperblock(id string): Obtiene SuperBlock y Partition para operaciones EXT2/EXT3.

Descripción de los Comandos Implementados

A continuación, se listan todos los comandos disponibles en la aplicación, incluyendo los de la Fase 1 y los nuevos de la Fase 2. Cada comando incluye una descripción, parámetros, ejemplos de uso, y su efecto sobre las estructuras internas.

Comandos

MKDISK

- **Descripción**: Crea un archivo .mia inicializado con ceros.
- **Parámetros**:
    - -size (obligatorio): Tamaño.
    - -unit (opcional, default M): B, K, M.
    - -fit (opcional, default FF): BF, FF, WF.
    - -path (obligatorio): Ruta.
- **Ejemplo**: mkdisk -size=10 -unit=M -path=/home/disco.mia
- **Efecto**: Usa utils.ConvertToBytes para calcular mbr_tamano y escribe el MBR con dsk_fit.

RMDISK

- **Descripción**: Elimina un archivo .mia.
- **Parámetros**:
    - -path (obligatorio): Ruta.
- **Ejemplo**: rmdisk -path=/home/disco.mia
- **Efecto**: Elimina el archivo sin modificar estructuras internas.

FDISK

- **Descripción**: Crea particiones primarias, extendidas o lógicas.
- **Parámetros**:
    - -size (obligatorio): Tamaño.
    - -unit (opcional, default K): B, K, M.
    - -path (obligatorio): Ruta.
    - -type (opcional, default P): P, E, L.
    - -fit (opcional, default WF): BF, FF, WF.
    - -name (obligatorio): Nombre.
- **Ejemplo**: fdisk -size=300 -unit=K -path=/home/disco.mia -name=Part1
- **Efecto**: Usa utils.ConvertToBytes y actualiza el MBR o agrega EBRs.

MOUNT

- **Descripción**: Monta una partición en memoria.
- **Parámetros**:
    - -path (obligatorio): Ruta.
    - -name (obligatorio): Nombre.
- **Ejemplo**: mount -path=/home/disco.mia -name=Part1
- **Efecto**: Usa utils.GetLetterAndPartitionCorrelative para generar Part_id y lo almacena en stores.MountedPartitions.

MOUNTED

- **Descripción**: Lista particiones montadas.
- **Parámetros**: Ninguno.
- **Ejemplo**: mounted
- **Efecto**: Lee stores.MountedPartitions y muestra los IDs.

MKFS

- **Descripción**: Formatea una partición con EXT2 y crea users.txt.
- **Parámetros**:
    - -id (obligatorio): ID de la partición.
    - -type (opcional, default full): full.
- **Ejemplo**: mkfs -id=671A
- **Efecto**: Usa stores.GetMountedPartitionSuperblock para ubicar la partición y escribe SuperBlock, Bitmaps, Inodos y Bloques.

CAT

- **Descripción**: Muestra contenido de archivos.
- **Parámetros**:
    - -fileN (obligatorio): Rutas de archivos.
- **Ejemplo**: cat -file1=/home/a.txt
- **Efecto**: Lee inodos y bloques de archivo con stores.

LOGIN

- **Descripción**: Inicia sesión (ahora manejado por la interfaz gráfica).
- **Parámetros**:
    - -user (obligatorio): Usuario.
    - -pass (obligatorio): Contraseña.
    - -id (obligatorio): ID.
- **Ejemplo**: login -user=root -pass=123 -id=671A
- **Efecto**: Valida en users.txt y actualiza stores.CurrentSession.

LOGOUT

- **Descripción**: Cierra sesión (ahora manejado por la interfaz gráfica).
- **Parámetros**: Ninguno.
- **Ejemplo**: logout
- **Efecto**: Limpia stores.CurrentSession.

MKGRP

- **Descripción**: Crea un grupo (solo root).
- **Parámetros**:
    - -name (obligatorio): Nombre.
- **Ejemplo**: mkgrp -name=usuarios
- **Efecto**: Modifica users.txt usando inodos y bloques.

RMGRP

- **Descripción**: Elimina un grupo (solo root).
- **Parámetros**:
    - -name (obligatorio): Nombre.
- **Ejemplo**: rmgrp -name=usuarios
- **Efecto**: Marca el grupo con ID 0 en users.txt.

MKUSR

- **Descripción**: Crea un usuario (solo root).
- **Parámetros**:
    - -user (obligatorio): Nombre.
    - -pass (obligatorio): Contraseña.
    - -grp (obligatorio): Grupo.
- **Ejemplo**: mkusr -user=user1 -pass=pass -grp=usuarios
- **Efecto**: Agrega usuario a users.txt.

RMUSR

- **Descripción**: Elimina un usuario (solo root).
- **Parámetros**:
    - -user (obligatorio): Nombre.
- **Ejemplo**: rmusr -user=user1
- **Efecto**: Marca usuario con ID 0 en users.txt.

CHGRP

- **Descripción**: Cambia el grupo de un usuario (solo root).
- **Parámetros**:
    - -user (obligatorio): Nombre.
    - -grp (obligatorio): Nuevo grupo.
- **Ejemplo**: chgrp -user=user1 -grp=grupo2
- **Efecto**: Actualiza users.txt y registra la operación en el Journal (EXT3).

MKFILE

- **Descripción**: Crea un archivo con permisos 664.
- **Parámetros**:
    - -path (obligatorio): Ruta.
    - -r (opcional): Crea carpetas padre.
    - -size (opcional): Tamaño.
    - -cont (opcional): Contenido.
- **Ejemplo**: mkfile -path=/home/a.txt -size=15 -r
- **Efecto**: Usa utils.CreateParentDirs para crear inodos y bloques. Registra la operación en el Journal (EXT3).

MKDIR

- **Descripción**: Crea una carpeta con permisos 664.
- **Parámetros**:
    - -path (obligatorio): Ruta.
    - -p (opcional): Crea carpetas padre.
- **Ejemplo**: mkdir -path=/home/docs -p
- **Efecto**: Usa utils.CreateParentDirs para crear inodos y bloques de carpeta. Registra la operación en el Journal (EXT3).

REP

- **Descripción**: Genera reportes en Graphviz.
- **Parámetros**:
    - -name (obligatorio): Tipo (mbr, disk, etc.).
    - -path (obligatorio): Ruta de salida.
    - -id (obligatorio): ID.
    - -path_file_1s (opcional): Ruta para reportes file o ls.
- **Ejemplo**: rep -id=671A -path=/home/mbr.jpg -name=mbr
- **Efecto**: Usa stores.GetMountedPartitionRep para leer estructuras y generar reportes.

FDISK (ADD y DELETE)

- **Descripción**: Permite agregar, quitar espacio o eliminar particiones.
- **Parámetros**:
    - -size (obligatorio al crear): Tamaño.
    - -unit (opcional, default K): B, K, M.
    - -path (obligatorio): Ruta.
    - -type (opcional, default P): P, E, L.
    - -fit (opcional, default WF): BF, FF, WF.
    - -delete (opcional): fast, full.
    - -name (obligatorio): Nombre.
    - -add (opcional): Espacio a agregar/quitar (positivo/negativo).
- **Ejemplos**:
    - fdisk -delete=fast -name=Particion1 -path=/home/Disco1.mia: Elimina rápidamente la partición "Particion1".
    - fdisk -add=-500 -unit=K -path=/home/misdiscos/Disco4.mia -name=Particion4: Quita 500 KB a "Particion4".
- **Efecto**: Actualiza el MBR o EBRs. Si se usa -delete=full, rellena el espacio con ceros. Registra la operación en el Journal (EXT3).

UNMOUNT

- **Descripción**: Desmonta una partición del sistema.
- **Parámetros**:
    - -id (obligatorio): ID de la partición.
- **Ejemplo**: unmount -id=341A
- **Efecto**: Elimina el ID de stores.MountedPartitions y reinicia el correlativo de la partición a 0.

MKFS (FS)

- **Descripción**: Formatea una partición con EXT2 o EXT3, creando users.txt.
- **Parámetros**:
    - -id (obligatorio): ID de la partición.
    - -type (opcional, default full): full.
    - -fs (opcional, default 2fs): 2fs, 3fs.
- **Ejemplo**: mkfs -id=062A -fs=3fs
- **Efecto**: Escribe SuperBlock, Journal (si EXT3), Bitmaps, Inodos y Bloques. Registra la operación en el Journal (EXT3).

REMOVE

- **Descripción**: Elimina un archivo o carpeta y su contenido, si el usuario tiene permisos de escritura.
- **Parámetros**:
    - -path (obligatorio): Ruta.
- **Ejemplo**: remove -path=/home/user/docs/a.txt

- **Efecto**: Elimina inodos y bloques asociados. Si es una carpeta, elimina recursivamente solo los elementos con permisos de escritura. Registra la operación en el Journal.

EDIT

- **Descripción**: Edita el contenido de un archivo, si el usuario tiene permisos de lectura y escritura.
- **Parámetros**:
    - -path (obligatorio): Ruta del archivo.
    - -contenido (obligatorio): Ruta del archivo con el nuevo contenido.
- **Ejemplo**: edit -path=/home/user/docs/a.txt -contenido=/root/user/files/a.txt
- **Efecto**: Actualiza los bloques del archivo. Registra la operación en el Journal.

RENAME

- **Descripción**: Cambia el nombre de un archivo o carpeta, si el usuario tiene permisos de escritura.
- **Parámetros**:
    - -path (obligatorio): Ruta.
    - -name (obligatorio): Nuevo nombre.
- **Ejemplo**: rename -path=/home/user/docs/a.txt -name=b1.txt
- **Efecto**: Actualiza el nombre en el FolderBlock padre. Registra la operación en el Journal.

COPY

- **Descripción**: Copia un archivo o carpeta y su contenido a otro destino, si el usuario tiene permisos de lectura y escritura.
- **Parámetros**:
    - -path (obligatorio): Ruta origen.
    - -destino (obligatorio): Ruta destino.
- **Ejemplo**: copy -path=/home/user/documents -destino=/home/images
- **Efecto**: Crea nuevos inodos y bloques en el destino, copiando solo los elementos con permisos de lectura. Registra la operación en el Journal.

MOVE

- **Descripción**: Mueve un archivo o carpeta a otro destino, si el usuario tiene permisos de escritura.
- **Parámetros**:
    - -path (obligatorio): Ruta origen.
    - -destino (obligatorio): Ruta destino.
- **Ejemplo**: move -path=/home/user/documents -destino=/home/images
- **Efecto**: Actualiza las referencias en los FolderBlock padre e hijo, sin copiar datos si está en la misma partición. Registra la operación en el Journal.

FIND

- **Descripción**: Busca archivos o carpetas por nombre, soportando caracteres especiales (? y *).
- **Parámetros**:
    - -path (obligatorio): Ruta de inicio.
    - -name (obligatorio): Nombre a buscar.
- **Ejemplo**: find -path=/ -name=?.*
- **Efecto**: Recorre recursivamente el sistema de archivos y muestra las coincidencias.

CHOWN

- **Descripción**: Cambia el propietario de un archivo o carpeta. Solo root puede cambiar cualquier archivo; otros usuarios solo sus propios archivos.
- **Parámetros**:
    - -path (obligatorio): Ruta.
    - -r (opcional): Aplica recursivamente.
    - -usuario (obligatorio): Nuevo propietario.
- **Ejemplo**: chown -path=/home -r -usuario=user2
- **Efecto**: Actualiza el I_uid del inodo. Registra la operación en el Journal.

CHMOD

- **Descripción**: Cambia los permisos de un archivo o carpeta.
- **Parámetros**:
    - -path (obligatorio): Ruta.
    - -ugo (obligatorio): Permisos UGO (por ejemplo, 764).
    - -r (opcional): Aplica recursivamente.
- **Ejemplo**: chmod -path=/home -r -ugo=764
- **Efecto**: Actualiza el I_perm del inodo. Registra la operación en el Journal.

LOSS

- **Descripción**: Simula una pérdida del sistema EXT3, sobrescribiendo áreas críticas con ceros.
- **Parámetros**:
    - -id (obligatorio): ID de la partición.
- **Ejemplo**: loss -id=061Disco1
- **Efecto**: Sobrescribe el Bitmap de Inodos, Bitmap de Bloques, área de Inodos y área de Bloques con ceros.

RECOVERY

- **Descripción**: Recupera el sistema EXT3 a un estado consistente usando el Journal.
- **Parámetros**:
    - -id (obligatorio): ID de la partición.
- **Ejemplo**: recovery -id=061Disco1
- **Efecto**: Reaplica las operaciones registradas en el Journal para restaurar el sistema.

JOURNALING

- **Descripción**: Muestra las transacciones registradas en el Journal (visualizado en la interfaz gráfica).
- **Parámetros**:
    - -id (obligatorio): ID de la partición.
- **Ejemplo**: journaling -id=061Disco1
- **Efecto**: Devuelve las entradas del Journal (operación, ruta, contenido, fecha) para su visualización en el frontend.