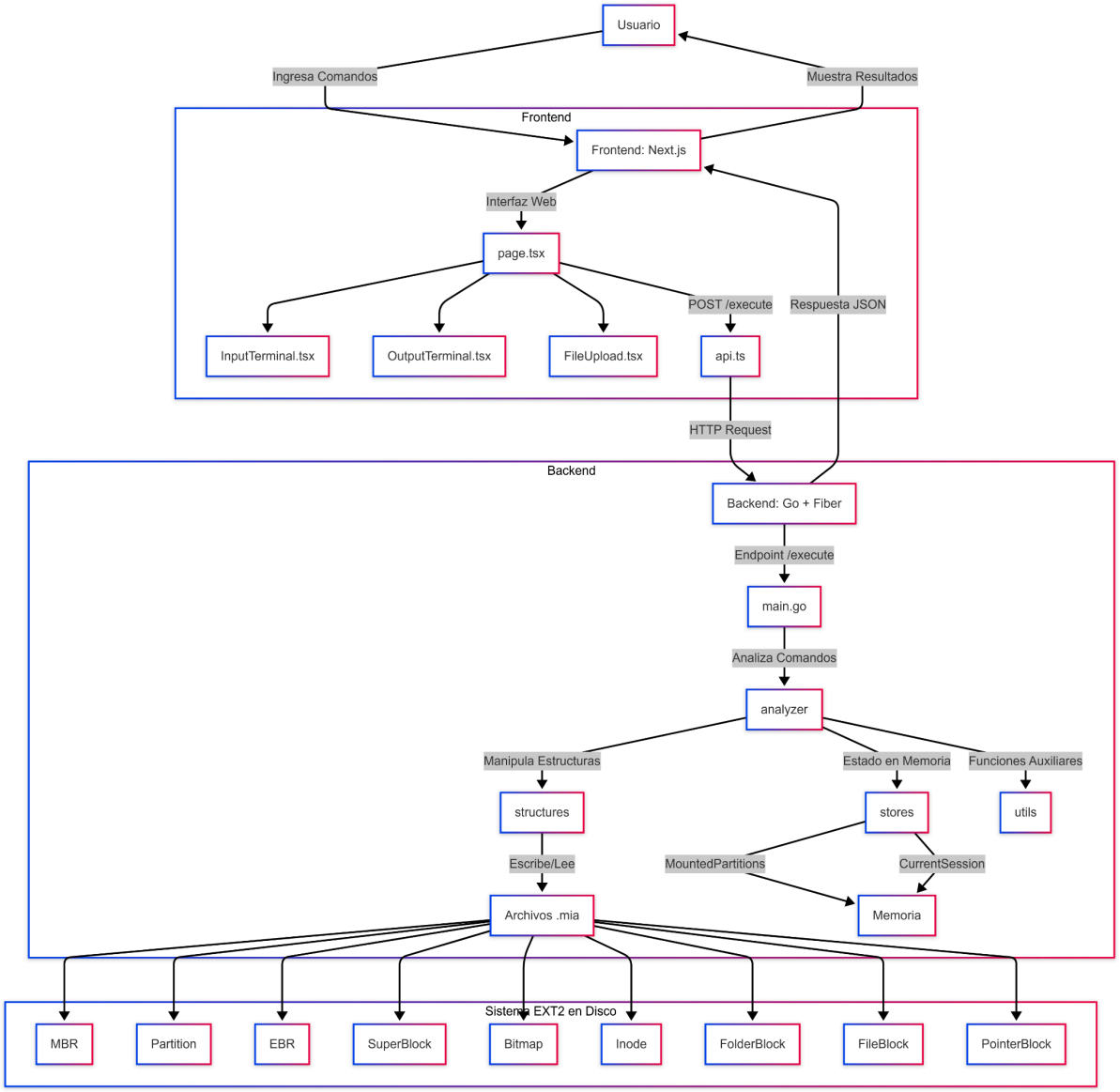
LABORATORIO MANEJO E IMPLEMENTACION DE ARCHIVOS - Sección B

Marcelo Andre Juarez Alfaro - 202010367

Manual Técnico - Proyecto 1: Sistema de Archivos EXT2

Descripción de la Arquitectura del Sistema

El sistema implementa una arquitectura cliente-servidor para simular un sistema de archivos EXT2, con un **backend** desarrollado en **Go** usando **Fiber** y un **frontend** en **Next.js**. El backend gestiona la lógica y las estructuras de datos en archivos .mia, mientras que el frontend ofrece una interfaz web interactiva. La comunicación se realiza mediante una API RESTful sobre HTTP. A continuación, se describe la arquitectura, sus componentes y su interacción, apoyada en el siguiente diagrama:



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). Maneja el estado (input, output, isLoading) y envía comandos al backend.
        - api.ts: Servicio que realiza solicitudes POST a http://localhost:3001/execute.
    - **Función**: Permite al usuario ingresar comandos o cargar scripts y muestra los resultados devueltos por el backend.
2. **Backend (Go + Fiber)**
    - **Tecnología**: Go con Fiber, un framework ligero para APIs RESTful.
    - **Estructura**:
        - main.go: Configura el servidor en el puerto :3001 con un endpoint /execute (POST). Parsea comandos y los pasa al analyzer.
        - analyzer: Interpreta y ejecuta comandos (e.g., MKDISK, FDISK), coordinando las operaciones sobre las estructuras.
        - structures: Define las estructuras EXT2 (MBR, SuperBlock, EBR, etc.) con métodos de serialización/deserialización.
        - stores: Gestiona el estado en memoria (MountedPartitions, CurrentSession).
        - utils: Ofrece funciones auxiliares (e.g., ConvertToBytes).
    - **Función**: Procesa comandos, manipula archivos .mia y devuelve resultados en JSON.
3. **Sistema EXT2 en Disco**
    - **Estructura**: Archivos .mia que almacenan las estructuras persistentes (MBR, Partition, EBR, SuperBlock, Bitmap, Inode, FolderBlock, FileBlock, PointerBlock).
    - **Función**: Representa el sistema de archivos EXT2 en almacenamiento físico, gestionado por el backend.

Interacción entre Componentes

- **Flujo**:
    1. El usuario interactúa con el frontend a través de InputTerminal.tsx o FileUpload.tsx.
    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.
    5. El backend devuelve un JSON con los resultados, que el frontend muestra en OutputTerminal.tsx.
- **Comunicación**: API RESTful (HTTP), con solicitudes como { "command": "mkdisk -size=10 -unit=M" } y respuestas como { "output": "Disco creado" }.

Explicación del Diagrama

El diagrama ilustra el flujo de datos y las relaciones entre componentes:

- **Usuario** → **Frontend**: Ingreso de comandos y visualización de resultados.
- **Frontend** → **Backend**: Solicitudes HTTP desde api.ts al endpoint /execute.
- **Backend** → **Sistema EXT2**: Manipulación de estructuras en archivos .mia mediante structures, con soporte de stores y utils.
- **Backend** → **Frontend**: Respuesta JSON que regresa al usuario.
  Los subgrafos destacan la organización interna del frontend (componentes React) y el backend (capas de procesamiento y almacenamiento).

Explicación de las Estructuras de Datos:

En esta sección se describen las estructuras de datos fundamentales utilizadas para simular el sistema de archivos EXT2 en el proyecto. Estas estructuras se almacenan en archivos binarios con extensión .mia, que representan discos virtuales. Además, se incluyen utilidades y estructuras auxiliares definidas en los paquetes utils y stores que facilitan la gestión de estas estructuras.

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 (int): Tamaño total del disco en bytes.
    - mbr_fecha_creacion (time): Fecha y hora de creación.
    - mbr_disk_signature (int): Identificador único (aleatorio).
    - dsk_fit (char): Tipo de ajuste del disco (B, F, W).
    - mbr_partitions (partition[4]): 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 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, sirviendo como base para el sistema EXT2 o EBRs.
- **Campos**:
    - Part_status ([1]byte): Estado de la partición:
        - 'N': Disponible (no usada).
        - '0': Creada.
        - '1': Montada.
    - Part_type ([1]byte): Tipo ('P' primaria, 'E' extendida).
    - Part_fit ([1]byte): Ajuste ('B' Best Fit, 'F' First Fit, 'W' Worst Fit).
    - Part_start (int32): Byte de inicio en el disco.
    - Part_size (int32): Tamaño en bytes.
    - Part_name ([16]byte): Nombre de la partición.
    - Part_correlative (int32): Correlativo para montaje (inicia en -1, asignado al montar).
    - Part_id ([4]byte): ID asignado al montar (e.g., 671A).
- **Organización**:
    - Cuatro instancias están embebidas en el MBR al inicio del archivo .mia.

    - Las particiones primarias ('P') alojan el sistema EXT2, mientras que las extendidas ('E') contienen EBRs para particiones lógicas.
- **Métodos**:
    - **CreatePartition(partStart, partSize int, partType, partFit, partName string)**:
        - Inicializa una partición con estado '0' (creada), asignando Part_start, Part_size, Part_type, Part_fit, y Part_name.
        - Usado por FDISK para definir nuevas particiones en el MBR.
    - **MountPartition(correlative int, id string)**:
        - Cambia el estado a '1' (montada), asigna Part_correlative y Part_id.
        - Invocado por MOUNT para registrar la partición en memoria.
    - **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.

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.

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 (e.g., 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(file *os.File, offset int64)**:
    - Escribe el EBR en el archivo .mia en la posición offset usando binary.Write en formato Little Endian.
    - Usado por FDISK al crear particiones lógicas.
  - **Deserialize(file *os.File, 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 para acceder a particiones lógicas.
  - **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 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, incluyendo el conteo de recursos, su estado y las posiciones de otras estructuras (bitmaps, inodos, bloques).

- **Campos**:
    - s_filesystem_type (int): Identificador del sistema (2 para EXT2).
    - s_inodes_count (int): Total de inodos, calculado según el tamaño de la partición.
    - s_blocks_count (int): Total de bloques (triple de inodos).
    - s_free_inodes_count (int): Inodos libres.
    - s_free_blocks_count (int): Bloques libres.
    - s_mtime (time): Último montaje.
    - s_umtime (time): Último desmontaje.
    - s_mnt_count (int): Veces montado.
    - s_magic (int): Valor mágico (0xEF53).
    - s_inode_s (int): Tamaño del inodo.
    - s_block_s (int): Tamaño del bloque (64 bytes).
    - s_first_ino (int): Primer inodo libre (inicia en 2 tras crear users.txt).
    - s_first_blo (int): Primer bloque libre (inicia en 2 tras crear users.txt).
    - s_bm_inode_start (int): Inicio del bitmap de inodos.
    - s_bm_block_start (int): Inicio del bitmap de bloques.
    - s_inode_start (int): Inicio de la tabla de inodos.
    - s_block_start (int): Inicio de la tabla de bloques.
- **Organización**:
    - Escrito al inicio de la partición EXT2 (justo después del Partition.Part_start o EBR.Part_start).
    - Seguido por 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)**:
        - (Idéntico al de bitmaps.go, parece duplicado en el código). 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 (e.g., en CreateUsersFile).
    - **UpdateBitmapBlock(path string, blockIndex int32)**:
        - Actualiza el bitmap de bloques, usado al asignar nuevos bloques.
- **Uso**:
    - Creado durante MKFS, define la estructura del sistema EXT2.
    - Sus campos s_first_ino y s_first_blo se actualizan tras operaciones como CreateUsersFile para rastrear recursos libres.

Bitmap

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

- **Función**: Gestionan la asignación de inodos y bloques, permitiendo un seguimiento eficiente de los recursos disponibles en la partición EXT2.
- **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 inmediatamente después del SuperBlock en la partición 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).
        - Para inodos: Crea un buffer de s_inodes_count bytes, todos en '0', y marca los inodos 0 (raíz) y 1 (users.txt) como '1' si hay espacio suficiente.
        - Para bloques: Crea un buffer de s_blocks_count bytes, todos en '0', y marca los bloques 0 (raíz) y 1 (users.txt) como '1'.
        - Usa file.Seek para posicionarse en s_bm_inode_start y s_bm_block_start, y escribe los buffers.
    - **UpdateBitmapInode(path string, inodeIndex int32)**:
        - Actualiza un inodo específico en el bitmap, marcándolo como ocupado ('1').
        - Verifica que inodeIndex no exceda s_inodes_count y usa os.OpenFile para abrir el disco en modo lectura/escritura.
        - Se posiciona en s_bm_inode_start + inodeIndex y escribe '1'.
    - **UpdateBitmapBlock(path string, blockIndex int32)**:
        - Similar a UpdateBitmapInode, pero para bloques, posicionándose en s_bm_block_start + blockIndex.
- **Uso**:
    - Durante el formateo inicial (MKFS), se crean con CreateBitMaps.
    - Se actualizan dinámicamente con UpdateBitmapInode y UpdateBitmapBlock al crear archivos o carpetas (e.g., MKFILE, MKDIR).

Inode

El **Inode** describe un archivo o carpeta en EXT2.

- **Función**: Almacena metadatos y apuntadores a bloques.
- **Campos**:
    - i_uid (int): UID del propietario.
    - i_gid (int): GID del grupo.
    - i_size (int): Tamaño en bytes.
    - i_atime (time): Último acceso.
    - i_ctime (time): Creación.
    - i_mtime (time): Última modificación.
    - i_block (int[15]): Apuntadores (12 directos, 1 simple, 1 doble, 1 triple).
    - i_type (char): Tipo (1 archivo, 0 carpeta).
    - i_perm (char[3]): Permisos UGO (e.g., 664).

- **Organización**: En la tabla de inodos, apunta a bloques gestionados por utils.SplitStringIntoChunks para contenidos.

Folder Block

El **Folder Block** almacena entradas de directorios.

- **Función**: Contiene nombres y apuntadores a inodos.
- **Campos**:
    - b_content (content[4]):
        - b_name (char[12]): Nombre.
        - b_inodo (int): Apuntador al inodo.
- **Organización**: Tamaño de 64 bytes, usado en inodos de carpetas.

File Block

El **File Block** guarda contenido de archivos.

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

Pointer Block

El **Pointer Block** gestiona apuntadores indirectos.

- **Función**: Contiene 16 apuntadores a otros bloques.
- **Campos**:
    - b_pointers (int[16]): 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 del alfabeto 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 File Block.

Estructuras en stores

El paquete stores maneja persistencia y estado en memoria:

- **MountedPartitions (map[string]string)**: Almacena particiones montadas (ID → ruta), actualizado por MOUNT.
- **Session**: Estructura para sesiones activas con ID, Username, UID, y GID, gestionada por LOGIN y LOGOUT.
- **GetMountedPartitionRep(id string)**: Recupera MBR y Superblock para reportes como REP.
- **GetMountedPartitionSuperblock(id string)**: Obtiene Superblock y Partition para operaciones EXT2.

Descripción de los Comandos Implementados

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 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 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.
- **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.
- **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.

MKFILE

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

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.

REP

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