Kickoff QRFS

Paula Kvist

Sarah Quesada

Proyecto 2

Sistemas Operativos

Semestre ii, Noviembre, 2025

Introducción

En el presente proyecto a implementar, vamos a abordar el desafío de crear un sistema de archivos funcional en el espacio de usuario para Linux. La característica fundamental es que se va a utilizar un conjunto de imágenes de códigos QR como medio de almacenamiento físico. En lugar de discos magnéticos o memoria flash, toda la estructura del sistema de archivos se va a encontrar codificada en estas imágenes.
La implementación se va a realizar en Rust como lenguaje de programación, aprovechando su enfoque en la seguridad de memoria y concurrencia. La interacción con el kernel del sistema operativo se va a gestionar a través de la biblioteca FUSE, lo que nos va a permitir registrar QRFS como un sistema de archivos válido sin necesidad de modificar el kernel.

Estrategia de Solución:
Nuestra estrategia se basa en un diseño inspirado en los sistemas de archivos Unix tradicionales, adaptado a las particularidades del almacenamiento en códigos QR.

El plan se divide en lo siguiente:

- Representación Física (Imágenes QR): El "disco" de nuestro sistema de archivos será una carpeta que contiene una serie de archivos de imagen (ej. 001.png, 002.png, ...). Cada imagen es un "bloque" físico de nuestro sistema.
- Estructura Lógica: Definiremos una estructura lógica clara que se mapeará sobre estos bloques físicos:

1. Superbloque (Superblock): Ocupará el primer código QR (o una porción de él). Contendrá metadatos globales: un número mágico para identificarlo como un FS de QRFS, el tamaño total del FS (en número de QRs), el tamaño de cada bloque, y punteros a las demás estructuras de control.

2. Mapa de I-nodos (Inode Map): Una tabla donde cada entrada (i-nodo) describe un archivo o directorio. Contendrá información como permisos, tamaño, propietario, y lo más importante, una lista de los bloques de datos (los otros QR) que contienen la información del archivo.
3. Mapa de Bloques de Datos (Data Block Bitmap): Una estructura de datos simple que nos indicará qué códigos QR están en uso y cuáles están libres para almacenar nuevos datos.
4. Bloques de Datos (Data Blocks): El resto de los códigos QR, que almacenarán el contenido real de los archivos.

- Proceso de Lectura/Escritura:

1. Escritura: Cuando se escriba un archivo, sus datos se dividirán en fragmentos del tamaño de la capacidad de un código QR. El sistema buscará QRs libres usando el Bitmap, los marcará como usados, escribirá los datos codificados en ellos y actualizará el i-nodo del archivo con las "direcciones" de esos QRs.
2. Lectura: Para leer un archivo, el sistema consultará su i-nodo para obtener la lista de QRs que lo componen, los leerá en orden, decodificará la información y la entregará al usuario.

Este diseño nos proporciona una base sólida y modular para construir las herramientas requeridas: mkfs.qrfs para inicializar estas estructuras, mount.qrfs para leerlas y exponerlas a través de FUSE, y fsck.qrfs para validar su integridad.

Ambiente de Desarrollo

Paso 2: Ambiente de Desarrollo

---

2. Ambiente de Desarrollo

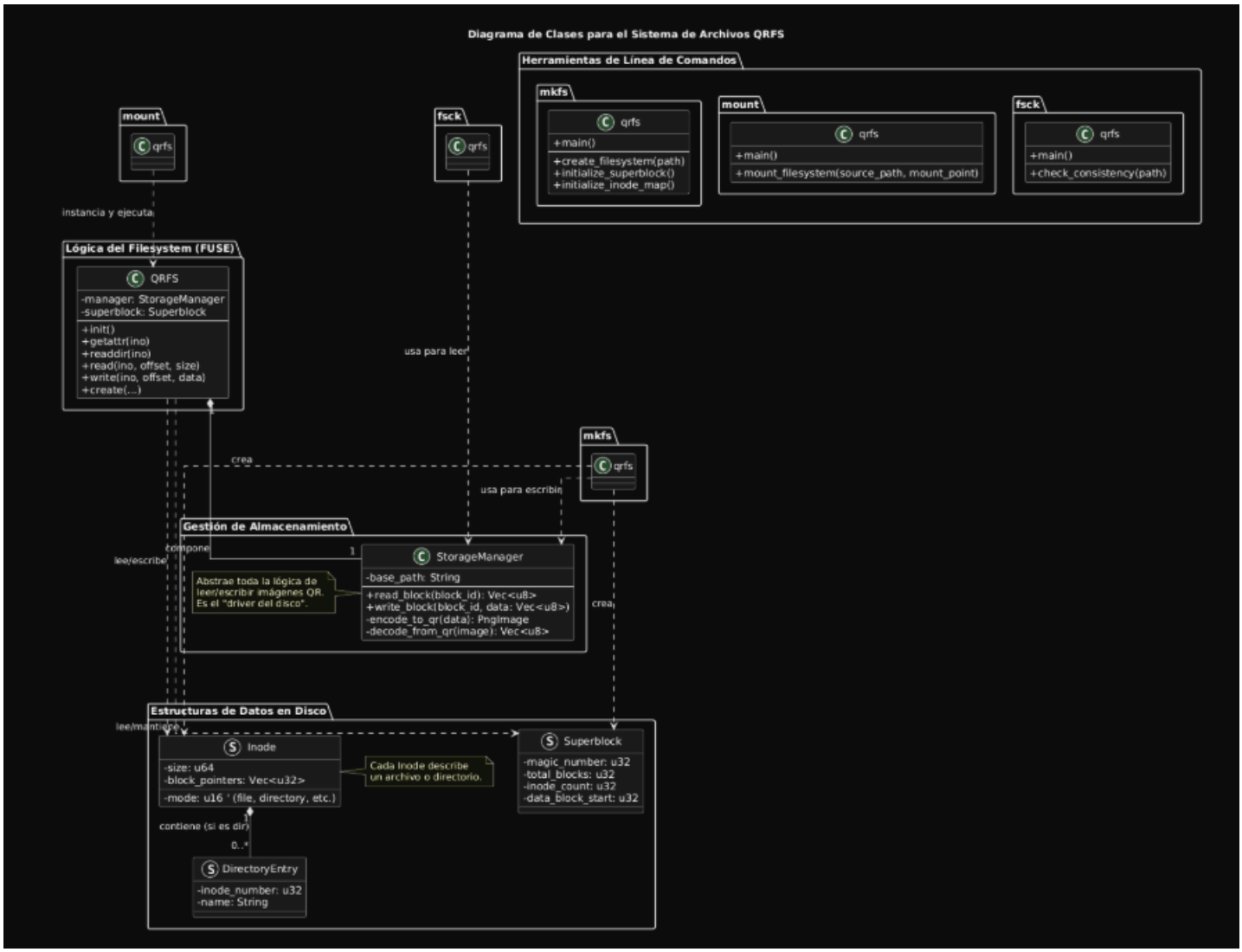
- **Lenguaje de programación:** Rust
- **Entorno de Desarrollo Integrado (IDE):** Visual Studio Code (VS Code).
- **Sistema Operativo y Dependencias del Sistema:**
  - **Sistema Operativo:** GNU/Linux (distribución recomendada: Ubuntu 22.04 LTS o similar).
  - **Dependencia Clave:** La biblioteca libfuse. Para compilar el proyecto, será necesario instalar los archivos de desarrollo correspondientes.
- **Bibliotecas Principales de Rust (Crates):**
  Cargo gestionará la descarga e integración de las siguientes bibliotecas esenciales:
  - **fuser:** Un binding de alto nivel y seguro para la biblioteca FUSE de C. Será el núcleo de nuestra implementación del sistema de archivos, permitiéndonos manejar las operaciones como read, write, getattr, etc.

    - **qrcode:** Una biblioteca para la generación de códigos QR a partir de datos binarios. Se utilizará para codificar los bloques de nuestro sistema de archivos en imágenes.
    - **image:** Una biblioteca para la codificación y decodificación de imágenes. La usaremos para guardar los códigos QR generados en archivos de formato PNG.
    - **serde:** (Opcional pero recomendado) Un framework para serializar y deserializar estructuras de datos de Rust de manera eficiente. Lo podríamos usar para convertir nuestras estructuras de control (Superbloque, I-nodos) a un formato de bytes para ser almacenados en los QR.
    - **bincode:** (Junto a serde) Para realizar una serialización binaria compacta, ideal para maximizar el uso del limitado espacio de un código QR.
- **Sistema de Control de Versiones :** GitHub

Control de Versiones

Repositorio

Diagrama UML



Explicación del diagrama:

1. **Herramientas de Línea de Comandos (mkfs, mount, fsck):**
   - Son los puntos de entrada que el usuario ejecutará desde la terminal.
   - mkfs.qrfs: Su responsabilidad es **crear** las estructuras de datos iniciales (Superblock, Inode para el directorio raíz) y usar el StorageManager para escribirlas en las imágenes QR.
   - mount.qrfs: Su única tarea es **iniciar** el sistema de archivos QRFS y pasárselo al sistema FUSE para que se monte.
   - fsck.qrfs: Usa el StorageManager para leer las imágenes QR y validar la consistencia de las estructuras de datos.
2. **Lógica del Filesystem (FUSE) (QRFS):**
   - Cada vez que el sistema operativo necesite hacer algo, va a llamar a una de las funciones de este struct (readdir, read, create, etc.).
   - Contiene una instancia del StorageManager para no tener que preocuparse de cómo se leen o escriben los QR.
3. **Gestión de Almacenamiento (StorageManager):**
   - Su única responsabilidad es manejar la interacción con el "disco físico" (la carpeta con imágenes QR).
   - Sabe cómo tomar un ID de bloque y convertirlo en un nombre de archivo (qr_codes/042.png).

- Contiene la lógica para llamar a la biblioteca de QR, codificar datos en una imagen PNG y guardar, o leer una imagen PNG y decodificar los datos que contiene.

4. **Estructuras de Datos en Disco (Superblock, Inode, DirectoryEntry):**
    - Representan la información que será serializada (convertida a bytes) y guardada directamente en los códigos QR.
    - Superblock: Contiene la metadata global del sistema de archivos.
    - Inode: Contiene la metadata de un archivo o directorio específico, y lo más importante, los punteros a los bloques de datos (QRs) que contienen su contenido.
    - DirectoryEntry: Es una estructura simple que se usa dentro del contenido de un Inode de tipo directorio para mapear nombres (como "archivo.txt") a números de i-nodo.