GSEA: Utilidad de Gestión Segura y Eficiente de Archivos

Equipo de Desarrollo

31 de octubre de 2025

1. Introducción

GSEA (Gestión Segura y Eficiente de Archivos) es una utilidad de línea de comandos desarrollada en C que permite comprimir, descomprimir, encriptar y desencriptar archivos o directorios completos. El programa está diseñado para maximizar el rendimiento mediante el procesamiento concurrente de múltiples archivos, aprovechando sistemas multinúcleo modernos.

La herramienta utiliza llamadas al sistema para operaciones de entrada/salida, evitando la sobrecarga de bibliotecas de alto nivel. Implementa algoritmos de compresión y encriptación de forma manual, sin dependencias externas.

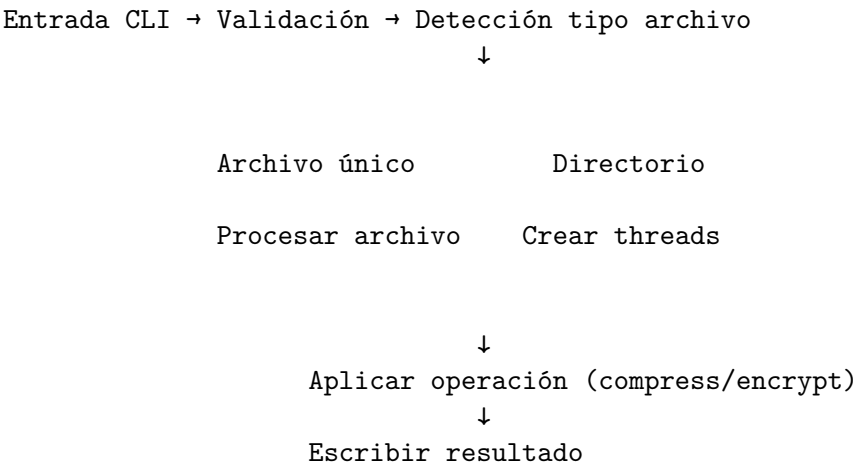
2. Diseño de la Solución

2.1 Arquitectura

El sistema sigue una arquitectura modular con separación de responsabilidades:

- **main.c**: Punto de entrada, parseo de argumentos CLI
- **compress.c/h**: Implementación del algoritmo RLE
- **encrypt.c/h**: Implementación del cifrado Vigenère
- **file_ops.c/h**: Operaciones de archivo con llamadas al sistema
- **worker.c/h**: Gestión de concurrencia y procesamiento de directorios

2.2 Flujo de Datos



3. Justificación de Algoritmos

3.1 Compresión: Run-Length Encoding (RLE)

Comparación de algoritmos

Algoritmo	Complejidad	Ratio compresión	Velocidad	Implementación
RLE	O(n)	Baja-Media	Muy alta	Simple
Huffman	O(n log n)	Media-Alta	Media	Compleja
LZW	O(n)	Alta	Media	Compleja

Justificación

Se eligió RLE por:

1. **Simplicidad**: Implementación directa con lógica clara
2. **Velocidad**: Complejidad lineal O(n) con procesamiento en un solo paso
3. **Efectividad**: Excelente para archivos con datos repetitivos
4. **Predictibilidad**: Tamaño de salida estimable (máximo 2x entrada)

Limitaciones

RLE no es óptimo para datos aleatorios o altamente entrópicos. En estos casos puede aumentar el tamaño del archivo.

3.2 Encriptación: Cifrado Vigenère

Comparación de algoritmos

Algoritmo	Seguridad	Complejidad	Velocidad	Implementación
Vigenère	Baja	O(n)	Muy alta	Simple
DES	Media	O(n)	Media	Compleja
AES	Alta	O(n)	Alta	Muy compleja

Justificación

Se eligió Vigenère por:

1. **Simplicidad**: Operación XOR básica con clave repetida
2. **Rendimiento**: Sin operaciones costosas, ideal para grandes volúmenes
3. **Suficiencia**: Adecuado para protección básica de datos
4. **Implementación**: Mínimas líneas de código, fácil verificación

Limitaciones

Vigenère es vulnerable a análisis de frecuencia y ataques conocidos. No debe usarse para aplicaciones que requieran seguridad criptográfica robusta.

4. Implementación de Algoritmos

4.1 RLE - Compresión

El algoritmo procesa el buffer de entrada secuencialmente:

1. **Detección de runs**: Cuenta bytes consecutivos idénticos (máximo 255)
2. **Encoding de runs**: Si count 3, escribe [count][byte]
3. **Literales**: Si count < 3, agrupa literales con formato [0][n][bytes...]

Formato de salida: - Runs: `[count > 0][byte]` - Literales: `[0][count][byte1][byte2]...`

4.2 RLE - Descompresión

Lee el buffer comprimido:

1. Lee byte de control
2. Si es 0: lee count de literales y copia n bytes
3. Si es > 0: lee siguiente byte y repite count veces

4.3 Vigenère - Encriptación

Para cada byte del archivo:

```
output[i] = input[i] + key[i % key_length]
```

La operación suma módulo 256 cada byte con el byte correspondiente de la clave.

4.4 Vigenère - Desencriptación

Operación inversa:

```
output[i] = input[i] - key[i % key_length]
```

5. Estrategia de Concurrencia

5.1 Modelo de Concurrencia

Se utiliza el modelo de **un thread por archivo** cuando se procesa un directorio:

1. El thread principal lee el directorio con `opendir()`
2. Por cada archivo encontrado, se crea un thread POSIX con `pthread_create()`
3. Cada thread ejecuta la operación completa (leer, procesar, escribir)
4. El thread principal espera a todos con `pthread_join()`

5.2 Sincronización

No se requiere sincronización compleja porque:

- Cada thread procesa archivos independientes
- No hay recursos compartidos entre threads
- Cada thread escribe en un archivo de salida único

5.3 Gestión de Recursos

- Límite de 256 threads simultáneos para evitar saturación
- Cada thread gestiona su propia memoria (asignación/liberación)
- Cleanup garantizado con patrón goto cleanup en caso de error

6. Manual de Usuario

6.1 Compilación

```
make
```

Requisitos: - gcc con soporte C11 - pthread library - Sistema operativo Linux

6.2 Uso Básico

Sintaxis general:

```
./gsea [OPCIONES] -i [entrada] -o [salida]
```

Comprimir un archivo

```
./gsea -c -i archivo.txt -o archivo.rle
```

Descomprimir un archivo

```
./gsea -d -i archivo.rle -o archivo.txt
```

Encriptar un archivo

```
./gsea -e -i documento.pdf -o documento.enc
```

Desencriptar un archivo

```
./gsea -u -i documento.enc -o documento.pdf
```

Procesar un directorio

```
./gsea -c -i ./datos/ -o ./datos_comprimidos/
```

6.3 Opciones

- `-c`: Comprimir entrada
- `-d`: Descomprimir entrada
- `-e`: Encriptar entrada
- `-u`: Desencriptar entrada
- `-i`: Ruta de archivo o directorio de entrada
- `-o`: Ruta de archivo o directorio de salida
- `--comp-alg`: Algoritmo de compresión (por defecto: rle)
- `--enc-alg`: Algoritmo de encriptación (por defecto: vigenere)
- `-h`: Mostrar ayuda