THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY

Department of Computer Science and Engineering

MINI PROJECT REPORT

Multi-threaded File Compressor and Decompressor (Using Run-Length Encoding - RLE)

Submitted in partial fulfillment of the requirements for
Operating Systems (PCS305)

Submitted by:

Name: [Your Full Name]

Roll No: [Your Roll Number]

Branch: [Your Branch]

Semester: [Your Semester]

Under the guidance of:

[Faculty Name]

Department of CSE
TIET, Patiala

Date: [Submission Date]

INDEX

Introduction

The 'Multi-threaded File Compressor and Decompressor' project is a C++-based system that implements the Run-Length Encoding (RLE) technique to compress and decompress files. RLE is a basic data compression algorithm that replaces consecutive repeated characters with a single instance of the character and a count of its repetition. This is especially effective on files with long sequences of repeated characters.

The primary aim of this project is to implement an efficient, thread-based version of RLE which divides the input file into chunks and processes them concurrently using multithreading. The benefits include reduced compression and decompression time and improved utilization of system resources. This makes the system suitable for real-time or large-file processing scenarios where performance is critical.

2. Working of the Project

 The program begins by reading a large text file (e.g., input.txt) for compression.

 It divides the file into smaller **fixed-size chunks** to enable parallel processing.

 Each chunk is processed **independently** using **multithreading** to speed up the compression task.

 The **Run-Length Encoding (RLE)** technique is applied to each chunk:

- Consecutive repeated characters are stored as a single character followed by a count.
- For example, aaaaa becomes a5.

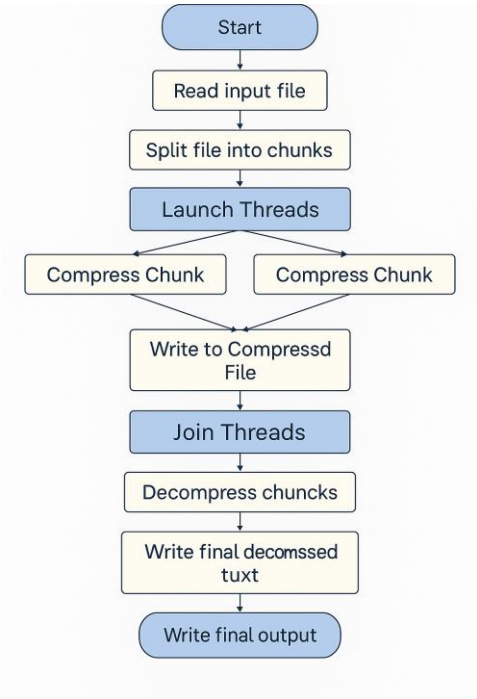
 The compressed output of each chunk is saved in a folder as separate .rle files (e.g., compressed_chunk_0.rle).

 Later, these .rle files are read and **decompressed in parallel** using threads.

 Decompressed chunks are stored individually as binary files.

 Finally, all the decompressed chunks are **merged sequentially** to reconstruct the original file.

 The result is saved in final_decompressed.txt, matching the input exactly.



3. File Descriptions

- main.cpp: Entry point that controls splitting, threading, compression, decompression, and merging.
- file_utils.h / file_utils.cpp: Contains RLE compression/decompression logic and file operations.
- compressor_program.exe: Compiled binary to run the compression stage.
- decompress_program.exe: Binary to run the decompression stage.
- Makefile: Automates the compilation of code.

4. Multithreading Logic

Multithreading is achieved using std::thread. Each file chunk is assigned to a thread for compression or decompression. This allows parallel execution of RLE on different data segments, reducing overall execution time.

Example:
threads.emplace_back(std::thread(compressRLE, input_chunk, output_file));

This line creates a new thread that compresses one chunk and saves the output.

Thread synchronization is not heavily required because each thread works on an isolated chunk. After all threads finish, the main thread handles merging.

Sample Output