Toni Au
Edmond Wu

CS214 Assignment 2: Procs vs. Threads Readme

Implementation:

- This program implements a variation of the RLE string compression algorithm in C. The algorithm essentially starts from the uncompressed string (extracted from the file) and iterates through it, using a counter variable to keep track of the number of identical, consecutive characters in the string. Each time a different letter character is encountered, the counter resets to 1 and that sequence of strings is compressed and added to a result string. The final compressed string is stored in an output file. Due to the algorithm requiring to iterate through the entire file string to check for repeating sequences, the runtime is essentially **O(*n*)**, with ***n*** being the length of the input string.
- **Multi-threaded approach:**
    - In the multi-threaded approach, the program splits the string into an array of strings with the size of the array determined by the number of parts to be split (specified by the user as a program argument). Then, for each string in the array, a thread is created to compress the string and write it to an output file. The program joins the threads, waiting for them to finish before terminating.
- **Multi-processing approach:**
    - In the case of multi-processing, via function “process_file_R”, the file is read, split into an array of strings where the size of the array is equivalent to the number of parts the user specified. Next, for each string in that array, a child process is forked. The actual compression of the string is done in a separate program, **compressR_worker_LOLS.c** which is called via *execvp*. *Execvp* allows us to access the other program while passing in the correct parameters for that child process. The required arguments for **compressR_worker_LOLS.c** are the part number, uncompressed string, and the input file name. After all parts are looped through and the output files are created, the parent process waits for all child processes to complete and the program finishes.

File setup:

- For modularization, several essential functions shared by both the multi-threading variant and the multi-processing variant are stored in a separate functions C file, which include:
    - **char* extract_file(FILE *file)**: extracts the file’s contents and stores them in a string
    - **char** split_string(char *string, int parts)**: splits an input string into a specified number of parts, with the split being done the way specified by the assignment instructions
    - **char* compress(char *string)**: function that runs the modified RLE compression algorithm
    - **int compressed_exists(char *file_name)**: checks if compressed output files exist
    - **void write_file(char *file_name, char *content)**: writes the compressed string into a specified file
- A makefile is created to enable ease of compiling the 2 separate programs. We used the C11 standard of the C language, so it’s recommended to use the makefile we created to avoid compilation errors (the program will not compile in C90).

Toni Au
Edmond Wu

Error handling:

- Errors are thrown on the following conditions:
    - Invalid input text file (either doesn't exist or don't have read access)
    - Compressed output files already exist (will have to delete them to re-compress them)
    - Invalid number of arguments
    - Number of parts to be split exceeds the number of characters in the file
- If the input file is empty or has only non-letter characters, the output file will be empty (as those are ignored by the compression algorithm)