Mike Botti
Zac Blanco
Assignment 2

CompressT & CompressR

CompressT and CompressR are both functions that use the LOLS algorithm to compress a text file into separate files based on input. Both functions have two arguments: a text file with at least 1 or more characters and the extension being .txt, and a number of splits that is less than or equal to the amount of characters inside of the text file provided. Both functions check to see if the inputs are valid and if so, compressT and compressR will generate N threads and processes respectively, where N is the number of workers specified by the parameters of the algorithm, that will each read the file at a given index until a certain length, and compress it using the LOLS algorithm. Each compressed section is written to a separate file, with the filename generated by a function inside the program. The filename follows the format, filename_txt_LOLS*n*, where *n* is the part number, if it exists. The generated files will appear in the same directory as the original uncompressed file.

Case Specifications

The filename does not end in *.txt*.

- The function will not produce any files.

The number of workers, N equals the number of characters in the string

- The string is simply split up into N files with one character each

The number of workers, N is greater than the number of characters in the string

- The function will fail and not produce any files. Nothing is returned. An error is printed

The number of workers is less than or equal to 0

- The algorithm will not create any files and print an error.

The file doesn't exist

- The no files are created and an error is printed

The file is empty

- No file(s) are created and an error is printed to the screen.

LOLS Algorithm

The LOLS algorithm is implemented in an iterative fashion where, given a string of N characters, will be compressed into "chunks". Each chunk represents a series of characters which is either represented as a single instance of that character, two consecutive instances of the character, or an integer of M digits preceding a single character denoting the length of the repeating series. In this fashion we are able to compress the amount of data that a file actually takes up by reducing the repeating bytes of the file using this compression scheme. Any characters which are not alphabetic are entirely ignored and can be treated as a loss of data.

Examples:
"a" becomes "a"
"aa" becomes "aa"
"aaa" becomes "3a"
"babaaa" becomes "bab3a"
"123bbb111ddddd" becomes "3b5d"
"1233124/.,/.,/.,/.,214" becomes ""

We can see from the third example that we get compression because we can now represent 3 characters as simply 2. We go on to show more examples of how the LOLS compresses different strings.

We should also note cases with non-alphabetic characters are ignored as it can be seen from the examples above.

Structure & Makefile

This project houses the following files:

- Makefile - makefile with targets to compile and test project
- lols.h - header file for lols.c
- lols.c - library of functions handling LOLS compression,
- compressT_LOLS.h
- compressT_LOLS.c - C file containing the thread worker method and initial method to run LOLS compression
- compressR_LOLS.h
- compressR_LOLS.c - C file containing function definitions for the multi-process version of LOLS compression
- compressR_worker_LOLS.h

- compressR_worker_LOLS.c - C file with main worker function for LOLS compression. MUST be compiled into a file named ./comp_proc.bin which resides in the same directory as the executable that wishes to use the process version
- test_lols.c - File containing unit tests for methods in lols, compressT and compressR files

Makefile Targets:
- all - compiles the 'make lols' (alias for lols target)
- clean - deletes all traces of lols compressT and compressR and compiled objects
- test - runs a large set of unit tests from 'test_lols.c'
- debug - compiles an debuggable version of compressT compressR and other files
- lols - compiles test_lols, compressR and compressT and all other necessary files. compressR_worker_LOLS is compiled to *compr_proc.bin*

Known Issue

There is one known issue with our program that occurs when using the valgrind to check for memory errors.

The command 'valgrind --trace-children=yes' (or any variant thereof) results in failing tests for some reason due to valgrind replacing the first argument when "exec"ing by the name of the executable instead of the original argument. We cannot figure out why exactly this issue arises but not using the command --trace-children=yes will result in the program running fine and all tests will pass. Otherwise tests ran with 'make test' will also pass. This issue only occurs when using '--trace-children=yes' with valgrind.