README

Kevin Daini, Mark Hirons

Assignment 2

IMPORTANT!!!

If you do not use our makefile then you must compile compressR_worker_LOLS.c to an output file called "compressRworker" or the exec will fail in compressR_LOLS.c

compressR_LOLS and compressRworker

The main process compressR_LOLS opens the file specified by the file and determines how to divide the characters in the file across the specified number of pieces. The main process spawns a child compressR_worker_LOLS process for each piece of the file. The parameters passed in the exec function are the "compressWorker, filename, piece start index, piece length, piece number". Each worker process opens the file given by filename and reads only the part of the file given by the start index and the piece length. The worker process then calls the compress function on a copy of the section of the file it is assigned. When the compression is complete the workers save their compressed part of the file to a new file called filename_txt_LOLS("" or piece number if > 0).

compressT_LOLS

Our implementation that uses threads was very similar to our processes implementation. We did not need a worker file however, so all functions required are in the same file. The file takes in a filename and a integer argument for the amount of threads desired.

The string from the file is divided into as many parts as desired, and the same amount of threads are created. The arguments for each thread are stored in a struct that holds 4 strings as arguments, *filepath, index, size,* and *number.*

Each thread executes a pseudo "main" function named threadMain(). The struct is passed to the thread along with a function pointer to threadMain().

The threadMain function takes in the arguments from the struct. It then compresses the string and writes the compressed string to a file.

Compress Function

Takes in a char * and uses Length of Long Sequence encoding to compress the string. Returns the compressed string.

Non alphabetic characters are ignored and are not included in the compressed file in any form.

It iterates over the given string and counts the consecutive identical characters and creates the compressed string. When a non-alphabetic character is encountered, the loop skips over it but the count does not reset, allowing consecutive characters separated by non-alphabetic characters to be counted correctly.

Non-alphabetic characters were not an issue in our function, as they were previously treated the same as alphabetic characters. We wrote the function to compress any character of any type given in the string, and it would handle it properly. But since we were told to skip non-alphabetic characters, we had to modify it to do so.