ReadMe - Asst 2 David Alummoottil and Anirudh Balachandran

Program Description:

In this assignment, we created a modified RLE compression program, LOLS, which accepts two parameters, a text file and integer, the former containing the information to be compressed, and the latter indicating to the program how many parts to split the data into. The latter integer is known as number of parts/number of kids/number of threads because depending on if the program is using processes or threads, the data will be split up into more manageable chunks of data to be compressed. compressT_LOLS is the program using threads while compressR_LOLS and compressR_worker_LOLS uses processes. In order to test this program, we created a makefile. This makefile compiles and creates executables for each of the three source code files we have. The executable for compressT_LOLS.c and compressR_LOLS.c are the program names themselves (excluding the .c).

Processes Program:

Our LOLS program implemented with processes uses the power of using multiple processes to complete the compression faster than if just one core process was used. In our parent process, we first check some error cases such as checking if the file is readable and if the data is in a compressible format. Then, we fork a child process from the parent for each part that the user inputs. Once the children are starting to be created, the program jumps into the “worker” file using execvp, which sends a char** array with the following: the executable name (will work as long as you use our make file to compile all the programs because our executable for the worker must be indicated in the code), file name, number of parts, and the child number (starting at 0….numberofparts-1). Once in the worker file, we first read the file and determine the total size of the file (This total size of the file includes NON-alpha characters), which we use to calculate other values used by the child to access and read the section of the file it is responsible for compressing. Once the child obtains the string it is responsible for (which includes non-alpha characters), it creates a new string which then takes out any non-alphabetical characters. This new string which does not have any non-alpha characters is then sent to the compression function and the compressed string is returned and output to the current directory as a file.

Threads Program:

Our LOLS program implement with threads uses the concept of multiples threads in order to encode an input file according to the RLE algorithm. This program contain two global variables which stores the thread number and the number of parts the input file has to be separated into and encoded. After these values are initialized the main() has a for loop that creates threads based on the user input. Using pthread create, the thread is created and sent to the myThread() function which accepts the input file from main and does the split for each thread. At first it open the file and check if it is a valid file. The next check is to check if the file has any alpha characters and proceeds accordingly. After passing these error checks, the file is opened and the substring is taken for that particular thread. After getting the string, the program gets a new string which contain only alpha characters and send this new string to the encode function to be encoded according to the RLE algorithm. The encoded string is returned back to the function and it is written into the output file and outputted for the user to view.

Output:

The output for our program depends completely on the input file.

- If the total number of characters is completely divisible by number of parts then the output each file gets the same number of characters. If it is not divisible then the number of extra character are added to the first N files where n is the number of extra characters.
- If all the characters are non-alpha character it just outputs an error
- If some of the characters are non-alpha characters and some are alpha characters than the input file completely ignore the process/threads that contain these non-alpha characters and only encodes the processes/threads which contain alpha characters
- If the total number of parts exceed the total number of characters then an error is outputted
- If the number of parts is greater than the total number of alpha characters for a particular process, then it outputs an error.
- If total number of parts is equal to zero then user is informed of this

- If total parts is not a number, an error is output.

All the outputs are as described in the assignment file and all errors are gracefully handled and the program comes to an end if an error is encountered either for the entire program or a particular thread/process. Examples of all these outputs can be found in the test cases file.