CS 214: Systems Programming
Fall 2016
Assignment 2: Procs vs Threads (round 0)

Riddhish Pandya & Kendrew Shum

Design:

For this project, we had to implement the modification of the RLE compression algorithm: Length of Long Sequence compression, or LOLS. The goal is to take an input text file of any arbitrary length consisting of any combination of any type of characters and compress it concurrently using processes and threads into any number of files. The first thing that had to be done was to account for all the possible error cases that the program can encounter from a simple incorrect input from the command line, which could crash the program. We accounted for incorrect number of arguments, non-existent input files, empty files, compressing into more files than characters within the file, and incorrectly formatted inputs. The difficulty in the actual compression of the file is to come up with a design that allows for concurrency by also maintaining the integrity of the variables of each individual thread or process . To start, we first wrote the processes program: compressR_LOLS.c (parent) & compressR_workers_LOLS.c (child/worker). A char* array was created which would hold an individual process's variables: start_index(where to start compression), end_index(where to end compression), name of file to compress, and output file number. The motivation behind this was that these variables would be calculated by the parent process and provided to the child process so that each individual process had its own variables to work with. A loop would consistently calculate each processes variables and then fork() itself, and the child would then call the worker function providing it with the variables. The parent process waits for the child outside of the loop to ensure concurrency. The thread program is a little more complicated because the stack space in a given "child" thread is the same as all the others. For this reason a struct is created which holds the individual "child" thread's variables. The parent calls pthread() for each thread and passes in a new struct. After all the threads return, they are all joined.

Important Assumptions on Implementation:

Because of a lot of ambiguity from the assignment, we would like to clarify a few ways of implementation.

1. We only compress alphabetic characters, everything else is ignored.

2. POSIX standard is that every file ends with a newline character, thus, we subtract 1 from the amount of characters retrieved by ftell().

3. Non-empty files with no alphabetic characters will output blank files.

4. The initial output intervals are calculated at the beginning, which count non-alphabetic characters. Thus, output files are dependent on non-alphabetic characters in input file.

How to Run Program:

1. Run make on the command line → provided a Makefile because executable names are important since compressR_LOLS.c calls execvp() on an compressR_worker_LOLS.

2. Input is of the form:

Threads: "./compressT_LOLS.c <file to compress> <number of parts>"
Processes: "./compressR_LOLS.c <file to compress> <number of parts>"