Ian Pascoe

Jean-Yves Hervé

CSC 412

21 November 2018

Prog05 Report

The premise of this assignment is to learn how create a multithreaded program to process images, create a script that monitors a drop folder for specific file types, and use pipes to communicate data from a server process to other client processes.

In Version 1 of this assignment, we exercise two of these three objectives. The C programs that were created are multithreaded version of the originals that were to be created for Prog02. Although my programs for crop and rotate had decent functionality, I used Hervé's code as a starting point due to the modularity of his design and the precision in error handling along with some other minor aspects. Multithreading these programs was not that difficult. I created a struct for the data needed for each thread, this included the start row and end row of pixels for each thread to work on, which was key to allowing even parallelism using the threads. Some other adjustments that had to be made to the code was a thread function that the threads would perform. This function took on most of the lines of code that were initially in main, because the threads were to perform most of the calculations. This includes the crop function, rotate functions, and split functions, the only difference with these functions is that the threads only perform the crop, rotate, and split on their designated pixels, specified by start row and end row. The script for this assignment uses inotify-tools to monitor a drop folder for .tga images and .job files. I found documentation online that allowed inotifywait to only react to certain file types. Once found, images are moved to the Data folder and all .job files are broken into lines and the

tasks inside are performed, as specified in the assignment instructions. The script will not terminate until the user enters the exit command (Ctrl+C). As far as I know, the programs I have provided will not crash unless the input in .job files are completely incorrect.

In version 2, all three of the objectives of this assignment are exercised. The only thing that needed to be changed in the C programs was the argument handling. I completely removed the arguments on the command line to allow for the pipes (created in the script; will be talked about shortly) to communicate the data from the .job files to the programs. Also, I just added a while loop where the condition was always true to allow for the program to continuously run and look for standard input. The script does a lot of the work in this version. It compiles the programs, initializes the programs and creates the communication pipes needed for all three of the programs, checks to see if the arguments for the script are valid and exist, monitors the drop folder, communicates the tasks to the running processes via the pipes, commences shut down when given the command and finally cleans up all the extra files that are not needed after execution. I ran into issues killing all the processes at the end of the script, which was quickly resolved when I learned you can save the PID of the last command performed to a variable.

I had used many more tasks than provided in the tasks folder that i kept deleting when performing tests. I provide a few that will test the functionality of all of the C programs using various images, but I assure you there were more and it will work for most if not all tasks tested.