**CS334: Operating Systems**

**Spring 2021. Assignment 3**

**Due: Sunday, March 31st 2021 NLT 11:59 PM**

**Individual, 100 pts possible**

**In this programming assignment you will write a multi-threaded solution that will process large number of image files, extract the color histogram from each of the images and write the output color histograms into a text file. Your solution has to compile and run on UP’s Linux VDI.**

Description:

You started working for a startup that is harvesting images from web cams to detect when the camera sees something “unexpected” in the image – an anomaly. For example, the images from a security camera would report anomaly if there is an unexpected person in a scene of an empty warehouse, the weather cams would report anomaly if there is a tornado on the horizon in a scene of a cityscape etc. The anomaly detector program, that you are NOT implementing, will not be fancy: the anomaly is defined as an unexpected color detected in the image. This would be implemented as a difference between the two color histograms extracted from either a constant, non-changing scene image or extracted from a scene image with an unexpected object that would result in a sufficiently different color histogram. The detector will learn to differentiate the “normal” color histograms from the “anomalous” color histogram, again this is not your job.

Your job is the implement the data pre-processing, in other words your task is to process all available images from one web-cam. The web cam is located at Poker Flats, Alaska and extract a color histogram from each image. The camera points to straight up to the night sky and takes the images of the night sky every couple of seconds. The anomaly detection for this task would be to recognize if the night sky has aurora in it or not (also known as the northern lights, <https://en.wikipedia.org/wiki/Aurora>). I’m giving you the algorithm to extract the color histogram from a single image.

Your task is to write a multi-threaded driver that will read the image file names from a folder, then extract the color histogram from an image sent to a thread for processing and finally write the histogram into a text file. Your solution will allow for processing multiple images simultaneously using multiple lightweight threads, where a single thread processes one image. The number of threads is one of the command line parameters passed into the program. The output histograms are written into a single output file.

The CLI execution requirements for your solution is:

./hist <thread count> <image folder> <output file name>

Example: ./hist 1 png cenek\_histogram.txt

I provided a starter code with an implementation of a RGB color histogram extracted from a PNG image (<https://en.wikipedia.org/wiki/Portable_Network_Graphics>). A color histogram of an image file is defined as follows: each pixel (a dot of an image) has three color components [Red][Green][Blue]; each color component is an 8-bit unsigned char value in 0-255 range; the color histogram is simply a count of all [Red][Green][Blue] values for all pixels of the image. The resulting histogram array is 256+256+256 long, where the first 256 values of the array are the counts of [Red] color components from all image pixels, the array offset 256-512 are the counts of the [Green] color components and the last part 256 values of the array have the counts of the [Blue] color composts from the all image pixels.

For example a 10x10 all black image has 100 pixels, each black pixel has RGB values of [0, 0, 0]. This image will produce a histogram with the value of 100 at positions 0, 256 and 512. To convert the counts into a frequency histogram, the array of values is divided by the pixel count, which will normalize all values to range 0-1.

The histogram output file <output file name> will have one image histogram per line followed by ‘#’ and the source image file name. This means that each line will have 768 floats separated by a space followed by # and string. The line is terminated by the end of line character. Example:

0.064457 0.013934 … 0.003078 0.012530 # PF\_image2018-01-31 01:10:07.210028.png

Starter code has the color histogram calculation, it does not have a memory leak, it does not print to a file, and the output is not in the correct format as described above. Feel free to re-structure the starter code however you need to. To compile your code, you need to include the statically linked libraries provided in your starter code bundle:

gcc -o hist imgHistogram.c libpng16.a libz.a -lm

Implementation Requirements/Hints:

* You have to protect writing the extracted color histogram into a file, as there is only one file pointer/descriptor and many threads writing into the same file. If unprotected, this would likely produce a jumble of intermixed thread writes.
* The main program/thread will be farming out the image files to be processed by different threads. At no point, the number of threads can exceed <thread count>. Implement this logic by declaring a global semaphore that will be decremented by the main program/thread when a new worker thread is started and incremented by the worker thread when it is about to exit.
* You have to pass multiple arguments to the worker-threads. Declare a struct called params that will “package” the parameters so they can be send into the worker thread. The parameters have to include, the image file name to be opened and processed by the worker thread, the opened file descriptor of the output file, etc.
* As there is only one struct instance that is passed to the thread, you have to protect writing and reading the params struct, from the time the program/thread writes data to it, to the point a thread reads the variables out and stores them in the worker thread’s local variables.
* The program/thread should open the <image folder> directory of the image files to be processed. The week two in-class activities have a C code example that show how to iterate over the directory of files. Make sure you only use the .png files, no directories, hidden files, etc.
* The program/thread should open the <output file name> file for writing the output color histograms
* The program/thread should have a loop that iterates over the image file names, decrements the semaphore, locks and loads the params struct with the image file name and output file descriptor and create a worker thread to process the image.
* In the worker thread move the struct values to local variables and unlock the struct.
* After the worker thread is finished processing the image, don’t forget to free any memory you allocated and increment the semaphore.
* Don’t forget to protect writing of the color histogram into the output file as multiple threads will try to write to the same file. Do not use the same mutex, to lock the operation of writing the color histogram to the file, as the mutex you used to protect the write/read of the params struct. Why is this?
* The main will loop until all images are processed.
* Don’t forget to free all mallocked data structures – the structs and arrays etc. you created (run valgrind to check for memory leaks). This includes freeing the IPC constructs like mutex, semaphores etc.
* If the semaphore count is exhausted, block the main program thread from creating additional threads and wait until threads complete before resuming spawning new threads. Use a spin-lock if no more threads can be created.
* Be careful with image file names, they contain spaces, ‘:’ and ‘.’ in the file name.
* When you have everything working on the sample images, download the large archive for testing and profiling. Measure the time it takes to process all files in the large archive to profile the lapsed time for Write up: question 2. Instead of the original 35k images, I’m giving you only 5k or 10k of images to test your program’s throughput. Please let me know if you want to try to process all 35k images.

<https://upedu-my.sharepoint.com/:u:/g/personal/cenek_up_edu/EZwTu0Y9FoZHgghI6V5TlrcBOsUgqPz0jrmmlY39S4M2KQ>

Evaluation criteria:

Functionality is 80% of your grade. The code must run UP’s Linux VDI (or Ubuntu 20.04 which is the same as the OS on UP’s systems). If you developed your multi-threaded program on some other OS you have to port and test it to UP’s Linux VDI. If there are some special notes that I would have to read about compilation, execution etc., please note them in README.txt file. The rest of your grade (20%) will be based on clarity, readability and detail of your write-up.

What to turn in:

An archive (.zip) NAMED as follows: <your userid>\_HW3.zip containing following files:

1. The source code files \*.c and \*.h. Please do not include the .a libraries OR the sample image files.
2. makefile to compile the code and produce the following target: hist
3. the output histogram file named: <your user name>\_histogram.txt
4. A write-up in a pdf format with answers to the questions in the “Write up” section of this document.
5. A readme file if there are any specific instructions I need to know when compiling and running the file. I will not read the file readme, unless the code does not compile or run.

Write up:

1. Did you complete all required sections and assignment’s functional requirements. If not state what is successfully implemented and what is not.
2. Measure the performance of you code with varying number of threads. Measure the execution time and report it here. Use the “time” utility to do this task.

|  |  |  |
| --- | --- | --- |
| Threads | Run Time | Comments and Notes |
| 1 |  |  |
| 2 |  |  |
| 4 |  |  |
| 8 |  |  |
| 16 |  |  |
| 32 |  |  |
| 64 |  |  |
| 128 |  |  |
| 256 |  |  |

1. It might be useful to plot the above performance in a graph. What did you notice in terms of the performance? Why is the performance not linear?
2. How would you increase the performance of your code?

After completing this assignment, you should be able to answer the following questions:

1. What is the limiting factor in programs with a massive number of threads?
2. What are the protection mechanisms for threads and shared memory?
3. How to manage many threads without closing them and restarting them from scratch.