TCSS 422 — Computer Operating Systems
Winter 2015 — Homework Assignment 3

Due Date: Thursday, Feb. 5

Guidelines

Homework should be electronically submitted to the instructor by the end of the day on the due date. A submission link is provided on the course Canvas page for this assignment.

Assignment Description

This assignment is intended provide you with further experience in multithreaded programming. In this assignment you will implement a small multithreaded program that analyzes a set of image files. The I/O portion of the program (loading image in from disk) cannot benefit much from concurrency and so will be handled by just a single thread. The CPU intensive portion of the program (analyzing each image) can benefit from concurrency and so will be handled by multiple threads. Images are analyzed to identify the largest axis-aligned rectangle of uniform color within the image. This type of image "landmark" can be useful in more advanced algorithms that perform image searches and comparisons. Starter code for this assignment is available on the course Canvas page for this assignment.

Implementation Specifications

The provided `analyzer.c` source file contains a skeleton for the entry function you're to implement, `analyze_images_in_directory`. This function will need to create threads, invoke other function that you write, etc., to implement the functionality of the program. The provided `main.c` source file contains a simple driver that reads in two command-line arguments, passes them to `analyze_images_in_directory`, and then displays the results of the analysis as returned by that function.

```
struct ImageInfo * analyze_images_in_directory(int thread_limit,
                                               const char * directory,
                                               int * images_analyzed)
```

This function should return a new, dynamically allocated array of `ImageInfo` structures that hold the results of the image analyses. `ImageInfo`, defined in `analyzer.h`, records the path of an image, i.e., its location in the file system, and the coordinates of the top-left and bottom-right corners of a rectangle within the image. The parameter `thread_limit` specifies the maximum number of simultaneous threads performing image analysis. The parameter `directory` specifies the directory that should be searched for images to analyze. The parameter `image_analyzed` points to an integer that should be set to the total number of images that were analyzed (the number of valid elements in the returned array).

One thread should load images from disk while one or more additional threads analyze images that have already been loaded. The I/O thread will need to go through all the files in the specified directory and load only those that are images in BMP format (recognizable by the extension `.bmp`). Each loaded image will be handled by an individual analysis thread. This setup is very similar to a producer-

consumer problem with the I/O thread representing one producer and the analysis threads representing multiple consumers.

There should be no more than `thread_limit` analyzer threads at any one time. If there are loaded images that aren't being analyzed and there are less than `thread_limit` analyzer threads in existence, a new analyzer thread should be created. Once an image has been analyzed, it is no longer needed. You should not make any assumptions about the total number of images that will be analyzed or the relative speed at with the I/O thread and the analyzer threads will be executed.

The BMP file format is a (relatively) simple raster (pixel-based) image format. The image is a rectangle of some width and height, and the color of every pixel is described in the file. This file format is a binary format not a text format, i.e., it contains binary data not ASCII-encoded text. The details of this format can be found on http://www.wikipedia.org/wiki/BMP_file_format. There is no standard C library function for loading such a file, and you are prohibited from using third-party libraries or code. You must write the code to load this type of image file yourself. However, you do not need to support all the features this file format supports. Specifically, you only need to be able to load BMP files that use 24 bits per pixel and uncompressed pixel data. Three small BMP images are provided for initial testing purposes and a larger collection will be released later.

The analysis of each image consists of locating within the image the rectangle of pixels of uniform color that has the largest area. The coordinates of this rectangle are used in the array that is eventually returned by the `analyze_images_in_directory` function. If the image contains several such rectangles of equal area, the rectangle nearest the top of the image should be recorded (and if there are still several such rectangles, the rightmost rectangle should be recorded).

It's highly recommended that you complete this program incrementally, adding functionality in stages:
1) Iterate through the contents of a directory and identify BMP image files.
2) Load each image file.
3) After each image file is loaded, analyze it (basically a single-threaded version of the program).
4) Load and analyze images in separate threads.

Deliverables

The following items should be submitted to the instructor. Failure to correctly submit assignment files will result in a 10% grade penalty.

5) The completed `analyzer.c` source file.
6) Any new supporting files, e.g., additional source code, needed by `analyzer.c`.

Do not include any extraneous files, such as Eclipse IDE files, object files, or subversion files.