CS 350: COMPUTER SYSTEM CONCEPTS

PROGRAMMING ASSIGNMENT 3: (OS PART)
IMAGE SUBSAMPLING VIA CLIENT-SERVER COMMUNICATION
ANNOUNCED: APRIL 12, 2008
DUE DATE: APRIL 24, 2008 (IN CLASS, DOCUMENTATION & PROGRAM LISTINGS)
APRIL 27, 2008, 11:59 PM (ONLINE SUBMISSION)
MAXIMUM MARK: 100

This assignment is based on the previous project in Assignment 2.

1. OBJECTIVE

The aim of this assignment is to test our understanding of key concepts in operating systems, such as processes, threads, critical section problems, and problems in inter-process communication. We will use an image analysis task as a platform for considering these problems.

A copy of this document and other required files for the assignment can be found via the course web page, under assessment:
http://www.csee.wvu.edu/~adjeroh/classes/cs350/

2. GENERAL

Image analysis is generally a time consuming process. In some cases, a special machine with special computing resources may be used for the analysis. Depending on the type of task, the required image analysis task can however be performed independently on different parts of the image. When there are different processors available, we may be able to speed up the analysis by performing some of the required tasks in parallel.

In this assignment, we shall simulate a situation whereby we have **a client** that needs to do some simple image analysis task. We shall also assume that we have available, **multiple servers** that can perform the required task. A client process can then send its requests to the servers, and each server will perform the required analysis and return the results to the client. For simplicity, we just assume that the client and the servers are in the same machine.

3. THE PROBLEM –IMAGE SUBSAMPLING (AS DESCRIBED IN ASSIGNMENT 2)

Given an ***m´ m*** image **A**, the required task is to reduce the image to a smaller image **B** of size ***s´ s***. This type of reduction is usually performed during the stage of image and video compression. For instance, the MPEG compression scheme used for video provides a mode that can be used to produce this type of reduced image. For simplicity, we assume that ***m*** is a multiple of ***s***. You can also assume that the input image is always square (but this does not have to be the case).

The required reduction is performed using the following steps.
Define ***n=m/s***. For each ***n´ n*** block of the image, we compute the average pixel value, and the standard deviation of the values in the block. (Overlapping blocks are not allowed). The average value then becomes the value for the corresponding ***single*** pixel position in the smaller ***s´ s*** image. For a given block size, the standard deviation of the pixel values in each block gives us a rough indication of how well the reduced image will approximate the original image. The smaller the standard deviation the closer the reduced image will be to the original image.

The figure below shows an example result for a real image.

Original image	n=2	n=4	n=8	n=16	

4. THE CLIENT SIDE (PARENT PROCESS)

At the client, we have a process that needs to produce a reduced version of an original image. The process will have to partition the input image into a total of ***s***^{2} ***n´ n*** smaller image blocks and distribute the partitions to its children. Each child will act as an image analysis server. Assume we have *p* children, ($p \leq 10$), but see the bonus problem.

The parent will do the following:

i. Read the input image (**imageIn.ppm**)
ii. Partition the original image into ***s***^{2} smaller subimages (or blocks) each of size ***n´ n***
iii. Distribute the jobs (the image blocks) amongst the *p* child-processes.
iv. Wait for the children to return their results
v. Arrange the returned results to form the reduced image
vi. Write the reduced image to a file (**imageOut.ppm**)
vii. Calculate and printout the overall image mean (i.e. mean of the individual block averages) and the average standard deviation using the results returned by all the child processes.

You are to decide on the specifics of how the parent and its children will communicate and how the parent will distribute the tasks to its children. For instance, since the parent and child process will have copies of the variables at the time a child is created, the parent can communicate which image parts each child will handle by modifying some variables just before the child is created. Thus, the image data will generally be accessible to the child, which will then extract its own assigned part of the image (for instance based on an index number) and then pass the data to the server. Similarly, you will need to decide on how you want the child processes to return their results to the parent, for instance via pipes, or some other mechanism.

5. THE SERVER SIDE (CHILD PROCESSES)

The server's job is simple.

i. Receive the input image block from the client.
ii. Perform that image analysis task
iii. Send back the results to the client

The actual task that the server will do is to calculate two simple image statistics - the mean and standard deviation - using the image block passed to it by the client.

Notes:

Given a collection of numbers, $x_1, x_1, x_3, \ldots, x_n$ the mean and standard deviation are defined respectively as:

$$m = \frac{1}{n}\sum_{i=1}^{n} x_i \; ; \qquad s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n} (x_i - m)^2} \; .$$

You are free to use threads (rather than child processes) on the server side.

Your server-based image subsampling program should be called **imServer.c**

6. THE INPUT DATA (SAME AS IN ASSIGNMENT 2)

The input data will be an image in **ppm** (portable pixel format) to be provided by the user. (Some sample images are provided in the project directory). Also provided are two functions, one to read images in **ppm** format, and the other to create a **ppm** image from a given data. These functions are in the image library: **iplib2New.c,** available in **imageInfo** directory. A sample program for using the image library is given in the file: **ip03mainSample.c**. You are not required to create your own program for reading or writing image files. But there will be no penalty for doing so :-). However, if you use a different image format, you will have to be able to read from and write to such formats. Also, you do not need a complete understanding of how the image read and write functions work. All you need is to understand the interface to the functions – i.e. how to pass parameters to the functions. Assume that images are only gray-level images.

7. Example Command Line (Notice minor difference with Assignment 2)

shell$> imServer imageIn.ppm imageOut.ppm *a* *p*

The parameter *a* is the reduction factor, *p* is the number of child processes (the servers).
That is, the reduced image dimension will be $\frac{M}{a} \times \frac{N}{a}$, where *M, N* are the original image dimensions.

8. Programming Style (10 MARKS)

Marks will be given for good programming style, exception handling, error checking, and special considerations for improving the program beyond the stated requirement. Examples here will be handling arbitrary image dimensions, handling color images, etc. The maximum here will be **10 marks**.

Algorithm Design Phase. For the problem above, you are required to include a documentation of the algorithm you used to solve the problem, before you started to code. The algorithm will carry about **20% of the mark**, before we start to consider the programs. This is different from the 10% mark for programming style. Submit algorithm design phase as a text file, or a file in .ps, .pdf or Word format. Also, you are required to submit hard copies of your assignment in class as indicated above.

The algorithm design phase and the documentation (if any) should not be more than **4 pages**.

Your program should be in C, and should be able to run on a UNIX platform. The program should be called **imServer.c**.

9. Bonus Problem (5 MARKS)

For the bonus problem, we will check the effect of the number of processes (or number of threads) and the block size on our image sub-sampling program. For each of the following three values of reduction factor *a*, *a*=2,4,8,16 run the program for different values of *p*.

Use *p*=2,4,6,8,10,20, 40, 60, 80,100.

Using the **time command** on the command line, record in a table, the time required for each run of your program for each of the four values of *a*. This should a table with three 10 rows and 4 columns. Plot the graphs of the time recorded (three plots, one for each *a*). You can put the three plots in one graph. Based on your graphs and the table, comment on the effect of *p* and *a*, on the time required to perform the enhancement. How does the average of the block standard deviations vary with increasing *a*, and why? Does *p* have any effect on the average of the block standard deviations? Put your table, graphs and comments as part of the documentation.

10. Extra Bonus Problem (5 MARKS)

Repeat the above extra problem, but using processes (if you used threads originally, or with threads if you used processes originally), and record your results and plot the graphs. How does the time when using processes vary with those recorded when you use threads?

11. Requirements

Your program should be written in C, and should be able to run on a UNIX platform.

Please submit your assignment using the usual ***submit*** command.
http://www.csee.wvu.edu/~adjeroh/classes/cs350/assessment/submit.html

Please also submit a **hardcopy** of your program files, algorithm design, and other materials in class on/or before the due date.