CSCI 4061 Spring 2017 | Assignment 4: A Threaded Image Organizer

Due:

Due on Sunday 04/16/2017 by 11:59 PM (Online, no hard copy)

Purpose:

The purpose of this assignment is to understand multi-threaded programming in C using POSIX thread library along with signaling and synchronization methods.

Scenario:

Think of any photo gallery app or picture organizer program, for e.g., Google's Picasa, which manages image files on your desktop. To accomplish these tasks, the application should have the capability to browse through the user's directories, locate relevant files and maintain path information for all files thereby creating a "catalog."

Assignment:

You are required to write a multithreaded C program using standard GNU C Library functions **ONLY.** The program must take as input the absolute path to a directory. For e.g. the home directory for an CSE-LAB UNIX user and parses through the directory tree extracting file specific information for files with four types of extensions: JPG, PNG, BMP and GIF. The file information gathered is written into a HTML file. Thread logs must be captured in a separate log `file`.

Task: File Information

For every image file of the given type in the directory tree capture the following information tuple (Hint: you could use a struct data type) in the following order:

```
FileId, FileName, FileType, Size, TimeofModification, ThreadId
```

where **FileId**: inode number of the file in the directory, **FileName**: Name of the image file in the directory, **FileType**: Type of the file in the directory, **Size**: Size in bytes of the file, **TimeOfModification**: Time when this file was last modified, **ThreadId**: Unique identifier of the thread responsible for writing the aforementioned information of the current image file to the HTML file.

Implementation variants

As you would have already realized, you can scan through the directory using threads in many ways. This assignment expects two different implementations which varies on thread creation.

Variant 1: In the first variant referred to as argument v1, the thread creation is done at a directory level. As directories are organized into a tree like structure, your program should assign a thread to every directory in the directory tree. Each of these threads search for files with valid image extensions, viz. jpg, bmp, gif, and png, within their assigned directory. If a thread encounters a subdirectory within its directory, it must then create another thread to scan through the subdirectory and so on and so forth.

Variant 2: In this second variant referred to as argument v2, you must create a thread for each directory as in variant 1 and then four new threads, one per file format for this specific directory. Thus, there will be five threads per directory regardless of the fact that the directory has pictures of all formats or not or subdirectories.

Variant 3: The third variant referred to as argument v3, **is similar to variant 1**, except now you have to ensure **Level Order Traversal**. This means all the images at a given level of the directory tree must be processed before moving on to the next level. Images at the same level must be processed concurrently (you can create a thread for each image in a given directory). To accomplish this, you start with the root directory and process all its image files. Then you move on to the second level directories. Image files in all these level 2 directories must be completely processed before you move on to level 3 and so on. The resulting level order structure would be evident in the HTML created.

Task: HTML File Generation

Your program should create **a single HTML** file which contains all the information in a record. Individual worker threads write all the information to the HTML file. This HTML file would consist of a list of images where an image icon (an icon of size 100X100 irrespective of the actual size of the image) provides a hyperlink to the original image file. Below each image hyperlink, the information tuple should be clearly documented.

NOTE: HTML file is written by all threads (including main threads), therefore you need to do synchronization among them (e.g. use mutex, semaphores). You have to make sure your program generates a working HTML. Please find a partial example of the HTML file below.

Your HTML must be called **“catalog.html”** If an HTML file already exists in the output directory when your program begins to run or for each variant you can safely you can override the same catalog.html saved in the output directory.

```
<html><head><title>My Image Manager BMP</title></head><body> <a href="./dir/5.bmp">
<img src="./dir/5.bmp" width=100 height=100></img></a><p align= “left”>
FileId, FileName, FileType, Size, TimeofModification, ThreadId</p> <a
href="./dir/6.jpg">
<img src="./dir/6.jpg" width=100 height=100></img></a><p align=“left”> FileId,
FileName, FileType, Size, TimeofModification, ThreadId</p> ………… </body></html>
```

Task: Logging

The ‘**main’** program should capture the program status and print it out into the log file. This status information would be updated by the working threads using global variables with mutex locks or semaphores and would be used by the main thread to print out into the log. This information should include the number of directories traversed, number of files handled, and number of html files created. The status needs to be printed every 100 milliseconds by the main process/thread ONLY using a continuous timer signal to itself. Use ITIMER_VIRTUAL and time.h libraries to update the status by printing it out on the log file.

Although all threads will write for successful matches in their directories to the HTML file only the first manager thread or the main thread should write to the log file. When all worker threads have finished, the manager thread then writes the information to **catalog.log.**

Another point to be taken care of for this step: If the “**catalog.log**” file does not exist in the target directory, you must create one and subsequently append. Threads logs must be generated and written into the same log file for running both the variants. The Log file maintains a record of previous executions of the code too. Every time your program runs you should append the information to the respective log file.

NOTE: The main thread writes Log file, therefore you need to collect information in a synchronized fashion from working threads.

Your recitation notes should be a good place to get hints on timers!

The status information should be based on the following template:

```
------- Time 100 ms                         #dir 3 #files 11 --------
------
... other log information
------- Time 200 ms .                       #dir 5 #files 12 --------
```

The total run time of each variant means the difference in begin and end times of the respective user defined functions, which implement these variants. In case you prefer coding in a single main () function, the time difference can be calculated for the blocks of code that implement the individual variants.

Input and Output

Your programs should take arguments **EXACTLY** as follows:

```
./image_manager [variant_options] ~/[input_path]  /[output_dir]
```

```
For e.g: assign4/$ ./image_manager v1 ~/home/csciUser/Desktop/a4/input_dir /myOutput
```

The program should take as input the start point in the directory tree and the name of directory where the catalogues and log files are stored. The executable has to be named image_manager and should run from any directory in the directory tree of a user.

Variant Number

`v1` the program runs under the variation 1 mode

`v2` the program runs under the variation 2 mode

`v3` the program runs under the variation 3 mode

Your program should begin its search from the ~/[input_path] directory (travelling through all subdirectories) and should save the resulting html file ./[output_dir]

Sample Directory Tree, HTML file and Log File

Download a sample directory tree with valid images and other files here. A sample log file generated during the process is available here.

The respective log files of the variants will contain the time and date for current running of the program, total time of execution of the variant, total number of files of each of the four formats. Only the manager thread and not the worker threads will write to a log file. A sample is provided with the handout.

Valid Assumptions

- There are four fixed file formats namely jpg, png, bmp and gif that you must consider for this assignment. Any other file format is to be discarded even name variants like jpeg etc.
- Filenames will be in lower case only and may contain special characters like - or underscore.
- Filenames will have a maximum length of 128 characters
- Validity of a file as one of the file formats to be considered in the assignment depends only on its extension. You do not need to check whether a file with extension bmp is indeed a bmp file. It is understood that a bmp extension in the filename means that the file is a valid image file for this assignment.
- There are no soft / hard links in the directory tree. There are only files and directories!
- It is possible to have ZERO files of one or more type. Your program does not need to treat this case any differently
- A log file is always opened in append mode. Hence if the log file is present for a variant in the target directory, your program should append to it.
- You are not allowed to use fork, exec, find, system or any other shell command
- You do not need to move or convert any files in this assignment. All you need to do is record absolute paths to all files of interest. The HTML code takes care of the rest.
- All files paths will be UNIX file system paths only.

What you need to turn in

- The source code of your program which includes *.h and *.c files
- A README.txt describing how you analyzed the run times for the variants and compare their performance based on time and space requirements.
- A makefile must be provided to compile and generate the executable. The executable should be named image_manager and not image_manager.o
- Do not print any debug statements to stdout. You can use output.log for other messages.
- You must have only one catalog.log file and one catalog.HTML file.
- All files (Code + MakeFile + ReadMe) must be in a single directory and the directory's name should be your and your partners University Student IDs
- The Log and HTML files should be generated when the program is run and should be named as catalog.html and catalog.log only. They have to be written into out_dir.
- Submission must be gzip compressed tar balls (ending in .tar.gz) ONLY
- Your directory structure should look like:

```
<studentid1_studentid2>/Makefile
<studentid1_studentid2>/README.txt
<studentid1_studentid2>/image_manager.c
<studentid1_studentid2>/out_dir/catalog.html
<studentid1_studentid2>/out_dir/catalog.log
  <studentid1_studentid2>/out_dir/output.log
```

Grading

1. Functionality

a. Error handling (Total: 10 points)

- Wrong number of arguments to the executable
- Output Directory Generation if missing

b. HTML File (Total: 20)

- catalog.html file should be generated correctly and should open in the browser
- catalog.html file should show the thumbnails and the file information correctly
- The ordering of the images in the html is important to test correctness of your code. This is particularly true for variant 3 where the results must be in level order.

c. Logging (Total: 20)

- v1: catalog.log file should be similar to the format shown in the handout
- v2: catalog.log file should be similar to the format shown in the handout
- v3: catalog.log file should be similar to the format shown in the handout
- Similarly, the output.log should clearly list out the important steps. Please ensure you log all the important steps clearly providing relevant information. You are the best judge. Your code should properly handle error conditions for system calls and file / directory IO. Thread creation, thread crash, thread termination and synchronization present very important scenarios for error handling. Please pay due attention to it. Thread crashes, should they happen, are to be reported on the error logs in output.log.

2. Compilation and Execution (Total: 30 → v1: 10 points v2: 10 points v3: 10 points)

Most importantly, your code should compile and execute in a desirable manner. By this we mean, that your code should create threads, read through directories, find files, report them by writing them in a synchronized way to the HTML file and finally report the Logs. You will not be evaluated for performance metrics of your code so do not worry about your program running slower than your friend’s.

3. Syntax, Documentation and Readability of Code (10 points)

It is helpful, both for a programmer and an evaluator, that a program is written with modularity, variables and functions have meaningful names and that comments precede function and logic blocks in the program. This however does not mean, that your comments should dominate the source code. If you wish to describe your methods in detail, please use the ReadMe.txt file for it.

4. Compliance and MakeFile (10 points)

You should comply with the entity nomenclature as defined in this assignment description. For that matter, the executable, HTML and Log file names should be exactly as mentioned herein. The program should execute with the exact format as defined in the Synopsis section. The Log files and HTML files should comply with the structure given in their respective samples. There must be a working MakeFile tested on UNIX machines.

Resources and Hints

You might want to take a look at the following online turorial on thread programming in C;

http://www.llnl.gov/computing/tutorials/pthreads/

You also require an understanding of directory and file handling in C. It is advisable to go through dirent.h header file. A couple of good resources for this are listed below

http://www.cs.cf.ac.uk/Dave/C/node20.html

http://users.actcom.co.il/~choo/lupg/tutorials/handling-files/handling-files.html

The assignment is not as complex as it looks prima-facie. Do not hesitate to ask questions should you get stuck at any point. Lecture, recitation, csci4061-help, forum a teaching personnel are good resources too. Enjoy the work and do not worry about grades. It is always good to start early. Please note the distribution of the marks and plan accordingly.