ECE 459: Programming for Performance Assignment 1*

Patrick Lam

January 19, 2014 (Due: February 2, 2014)

In this assignment, you'll work with a program which requests a resource across the network. I've provided a single-threaded implementation which uses blocking I/O to get the resource. You will reduce the latency of this operation by sending out multiple requests simultaneously (to different machines). In part 1, you'll use pthreads to do this, while in part 2, you'll use nonblocking I/O.

Setup

After setting up your ssh key at `http://ecegit.uwaterloo.ca`, fork the provided git repository at `git@ecegit.uwaterloo.ca:ece459/1141/a1`:

```
ssh git@ecegit.uwaterloo.ca fork ece459/1141/a1 ece459/1141/USERNAME/a1
```

and then clone the provided files. (You can also download the provided files at `http://patricklam.ca/p4p/files/provided_a01.tar.gz`, but don't do that if you're in the course—it'll make submitting harder.)

You should do this assignment in Linux, as the provided `Makefile` was only tested on Linux and isn't very robust. Use a virtual machine at your peril. It might work OK since the program's bottleneck is the response time of the remote execution.

You may log into `ece459-1.uwaterloo.ca` with the ssh key that you set up at `ecegit`. Use your uwuserid with that ssh key. I'll be testing your solutions on that machine.

Assignment code walkthrough

You will find the file `paster.c` in the provided file. This code uses `libcurl` to fetch a set of `PNG` files from the network and `libpng` to paste the files together.

I've provided a web API which returns portions of a raytraced image. You can see this in a browser by visiting `http://machine:4590/render/image.png?x0=0&y0=0&w=50&h=50`, replacing `x0`, `y0`, `w` and `h` appropriately, and where `machine` is one of `patricklam.ca`, `berkeley.uwaterloo.ca` and `ece459-1.uwaterloo.ca`. The backing image is currently 640×480.

To be precise, the provided code fetches 640×50 strips of the image, puts them in an array, and then produces an output file, `output.png`, with the pasted-together image.

*r0: initial version; r1: add alternate nbio option; r2: add requirement for documenting benchmark results; r3: add requirement for fixing resource leak and instructions for ece459-1

Part 0: Resource Leaks (5 marks)

I inadvertently left a resource leak in the provided code. Resource leaks sap performance. Find it (valgrind helps), fix it, and document it in your report.

Part 1: Pthreads (50 marks)

Use the `pthread` library create a threaded version of the provided program. Your program should create as many threads as the `num_threads` variable (which reads the value from the `-t` command line option) and distribute the work between the 3 provided servers. Make sure all of your library (standard glibc, libcurl, and libpng) calls are *thread-safe* (for glibc, e.g. `man 3 rand` to look at the documentation).

We will look at your code to ensure that it uses `pthread` calls properly, and we will execute your code to verify that it produces the correct output. Code that doesn't compile will get 0 marks.

(10 points) In your report, describe how you know that your threaded code uses only thread-safe calls, with pointers to the appropriate documents, and why your code is free of race conditions.

Also, time your executions with the serial version and parallel version (take an average of 6 runs each) and discuss—how well does parallelization work?

Part 2: Nonblocking I/O (45 marks)

In this part, you will write a single-threaded version of `paster` which uses nonblocking I/O to request multiple versionf of the image simultaneously. You will need to use the `curl_multi` API as well as `select`. Once again, distribute the work between the 3 provided servers.

Your solution should *not* use pthreads. However, it should have multiple concurrent connections to servers open. In this case, the `-t` command line option indicates the number of connections to keep open at once.

Again benchmark your work and report comparative results. Discuss the performance of all three versions. Is it what you expected?

Alternate option. You may instead provide a client-side JavaScript solution which programmatically initiates multiple requests, integrates the results, and displays the resulting image. (You may not, for instance, just submit a webpage which contains a bunch of HTML `img` tags.) I'm thinking of something like a solution with `node-pngjs`. I will not provide information on how to accomplish this; you are on your own (but come talk to me if you're interested).

For extra coolness (but no marks): I will probably implement a `zoom` option on the web API. Support zooming in and out on the image. I think that would be a lot of fun!

Submitting

To submit, simply push your fork of the git repository back to `ecegit.uwaterloo.ca`. We will be marking `Makefile`, `src/paster.c`, `src/paster_parallel.c`, `src/paster_nbio.c`, and `report.pdf`. (You can modify the provided `report/report.tex` and create it with `make report`; do not submit a `doc` file!). Running `make` in the `assignment-01` folder should produce three files: `bin/paster`, `bin/paster_parallel`, and `bin/paster_nbio`.