ECE 252: Systems Programming and Concurrency Spring 2019

Lecture 33 — Starling City Vigilantes

Jeff Zarnett *2019-03-05*

In Class Exercise:

Background. Your name is Oliver Queen. After five years in hell, you have come home with only one goal: to save your city. Now others have joined your crusade. To them, you're Oliver Queen. To the rest of Starling City, you're someone else. You are something else. You are... the Green Arrow.

The League of Assassins has planted an explosive device in one of the buildings around town. There are too many buildings for you to search by yourself, but you have a team of your fellow vigilantes who will help you. It's your team, so you will need to coordinate the actions of your team members – telling them what areas they should search, and checking in with them as to whether they've found it. You are a hands-on individual, so you will also be searching in the meantime.

Either you or a member of your team will find the device eventually. Once it has been found, you can tell your team members they can stop searching. If you found the device, deal with it immediately. If someone else found it, they will tell you where it is so you can deal with it. Once it's disabled, the city is safe for another night and you can go back to the lair.

Primary Objective. The primary objective of this exercise is to practice using asynchronous I/O in a program.

Secondary Objective(s). This is an additional opportunity to practice with system calls, program design, . You may also improve your ability to work with version control (git) and gitlab.

Starter Code. The starter code can be found at `https://git.uwaterloo.ca/jzarnett/ece252/ece252-e4` – fork this repository to your own space. Set permissions for this repository to be private, but add the group for the course staff with read access so your code can be evaluated.

Submitting Your Code. When you're done, use the command `git commit -a` to add all files to your commit and enter a commit message. Then `git push` to upload your changes to your repository. You can commit and push as many times as you like; we'll look at whatever was last pushed. And check that you gave the course staff permissions!

Grading. Binary grading: 1 if you have made a meaningful attempt at implementing the code; 0 otherwise.

Description of Behaviour. The goal is to implement the program so that the following behaviour occurs:

- Each building needs to be searched. You will do some yourself as shown in the starter code (every 5th building is assigned to you). You can search one building at a time the normal way (with blocking reads), as is shown in the starter code.
- The majority of the work should be handed out to team members using POSIX AIO. And the AIO requests can all be started up-front, before you search any buildings yourself.
- For each building you're not going to search yourself you should create and enqueue a POSIX AIO request.
- For each building, you need to create an AIO control block and its associated buffer.
- Offset should be 0 in each control block.

- The event action when the AIO is complete should be to create a new thread; the new thread should run the function name of the team member who has been assigned that building.
- You have four team members helping you: Diggle, Speedy, Black Canary, and Arsenal. Each of them can be in only one place at a time, so it is best to distribute the work evenly between them.
- The argument in the event (`sigev_value.sival_ptr`) should be a pointer to the buffer representing the building (i.e., the same as the read buffer).
- Your program should not leak memory; be sure to destroy/deallocate anything initialized/allocated.
- There should not be any race conditions in your program either.

Hints & Debugging Guidance. Some general guidance is below. If you're having trouble, try running through these steps and it may resolve your problem. If you're still stuck you can ask a neighbour or the course staff.

- If you need to catch up on how AIO works then you can look at the lecture on the subject in the course notes or [SR13] chapter 14.5.3.
- If your program seems to hang, it may be because the `found_it` semaphore is never triggered.
- If you get double `free` alerts it may be because you are re-using a buffer inappropriately.
- To narrow down the source of a problem, you could assign all work to yourself (main) or to one team member; if the problem does not occur then concurrency may be an issue.
- Check the documentation for how functions work if you are unfamiliar with them (google is your friend!)
- Have you initialized all variables? It is easy to forget; `malloc` does not initialize the value...
- Is there a missing or extra `*` (dereference) on a pointer somewhere?
- Does every memory allocation have a matching deallocation?
- Check whether values are stack or heap allocated (this can cause problems).
- It may be helpful to put `printf()` statements to follow along what the program is doing and it may help you narrow down where the issue is.
- Don't be shy about asking for help; the TAs and instructor are here to help you get it done and will help you as much as is reasonable.

References

[SR13] W. Richard Stevens and Stephen A. Rago. *Advanced Programming in the UNIX Environment, Third Edition*. Addison-Wesley, 2013.