ECE 252: Systems Programming and Concurrency Spring 2019

Lecture 25 — The Time War

Jeff Zarnett *2019-07-06*

In Class Exercise: Gallifrey Stands!

Background. It is the last day of the Time War. The Daleks have launched their final assault on the Time Lords' home planet of Gallifrey. Ten million Dalek ships surround the planet and are bombarding it from orbit and sending landing parties. The War Doctor has stolen an ancient weapon called The Moment. It will wipe out the Daleks – but also Gallifrey, and everyone on it.

But there's another way. Through hundreds and hundreds of years' worth of calculations The Doctor can use stasis technology to freeze the whole planet of Gallifrey in a parallel pocket universe. The Daleks will be obliterated in the crossfire and it would look like they wiped each other out. This takes hundreds and hundreds of years of calculations and the power of every one of the thirteen incarnations' TARDISes.

This won't be easy – the timing has to be just right and the Doctors have to coordinate their actions and timing and work together to save Gallifrey. If their timing is wrong or if things are out of sync or important steps are skipped, the Doctors will fail and Gallifrey falls. If you can find and correct the problems in the program, then Gallifrey stands!

Primary Objective. The primary objective of this exercise is to identify and solve memory management and concurrency problems in the provided code.

Secondary Objective(s). This is an additional opportunity to practice concurrency and synchronization techniques. 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/ece252-1195/ece252-e3` – fork this repository to your own space. Set permissions for this repository to be private, but add the course instructor with developer 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 instructor 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 analyze and correct the program so that there are no longer any errors (that you can fix in your program) reported by Helgrind, and also no memory leaks/issues (that you can fix in your program) reported by Valgrind.

When you compile and run the starter code, the program may crash or hang when executed. Those are just two of the symptoms of problems present. There are a total of nine (9) different problems in the code to be resolved. Some will be found by the memcheck tool of Valgrind; others will be found by the Helgrind tool. If you read the error messages carefully, you will be able to identify the causes of the problems and fix them.

There will be some things reported by both tools that are beyond your ability to fix (because they are in libraries) or otherwise misreported. You'll need to examine the messages carefully and decide if it is something you can fix or not. If not, just leave it be. Generally speaking, errors reported by memcheck need to be fixed; those reported by Helgrind need to be examined more carefully.

If something is wrong, you may see "Gallifrey falls" as an output message at the end of your program. If everything goes well you will see the message "Gallifrey Stands!" at the end of the program, but just seeing that that is not enough. You also need a clean bill of health from Valgrind and Helgrind.

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 instructor.

- Check the documentation for how functions work if you are unfamiliar with them (google is your friend!)
- See the docs for Valgrind [Dev15].
- Read the valgrind error messages carefully: they usually tell you which function (i.e., which doctor) the problem occurs in. This helps you narrow it down
- 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

[Dev15] Valgind Developers. Valgrind tool suite, 2015. Online; accessed 24-November-2015. URL: `http://valgrind.org/info/tools.html`.