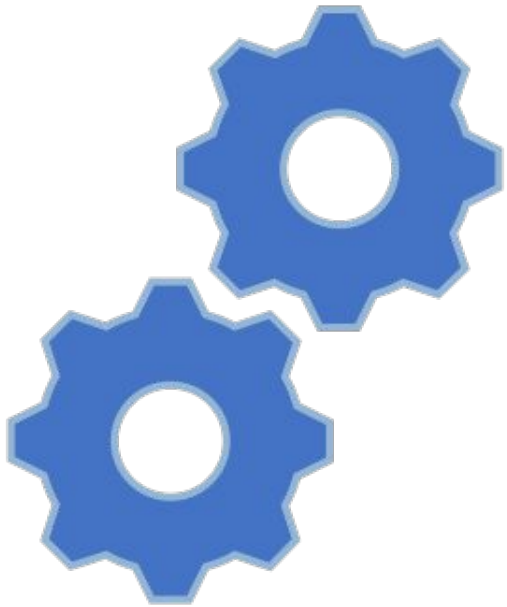
Go Debugging

Fall 2021



Overview

- Congrats on reaching the P1 checkpoint! 🏅
- Part A final is complicated
  - By the end, your implementation will have many moving parts
  - Our goal is to help you fix the bugs that inevitably arise
  - Debugging large projects like this is a crucial real-world skill!
- Purpose of this recitation:
  - Teach language-agnostic debugging skills
  - Teach Go-specific debugging skills
  - Outline expectations for questions asked on Piazza, Slack, or at Office Hours
- **CHECK OUT THE FAQ POST! @279**

Debugging Tip1: Logging

- Add log statements around points of inter-thread communications
  - **Channel -> understand buffered & unbuffered channels!**
  - Connection
  - ...
- Attach server/client IDs to each log line
- Create multiple log files, one for each thread
- Limit log output size for easy navigation
  - Convenient to use flags to enable/disable logging
  - No multi-level logging support in Go's standard library

Debugging Tip2: Race Condition

- Remember to **run with –race flag** and **read –race output** (it tells you the two specific go routines that are having data race!)
- Goland has a built-in debugger
- @68: Use the defer statement from the quick start in every test you'd like to leak test.
  - https://github.com/uber-go/goleak

Debugging Tip3: Read the Tests

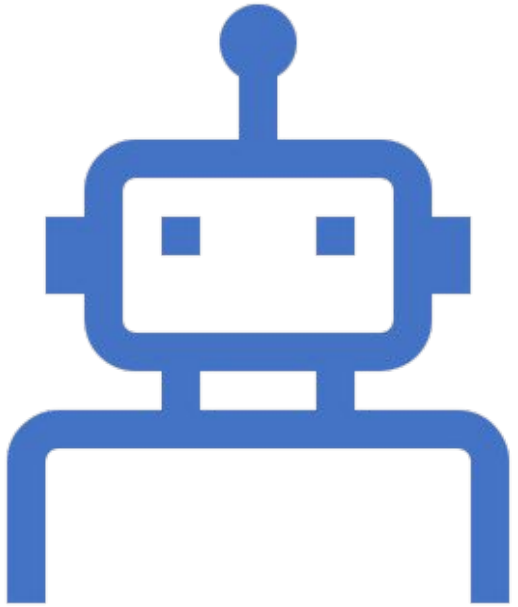
- Why read the tests programs?
  - Understand the expected behavior of the program.
  - The system specification is written in natural language and thus inherently **ambiguous** and prone to **misinterpretation**.
  - The test program is more precise.
- Prepare for Part B.
  - The public tests are simple. The hidden tests are brutal.
  - You will need to write some of your own tests for partB!

Golang Documentation

- Golang documentation can sometimes be cryptic, unhelpful, terse, or simply empty. However, sufficient digging can usually unearth the details you are interested in.
- A common question in p0 was: Why does my code stop at conn.Read()?
  Can you try to figure this out yourself? (Other than reading the handout)

Step1: Fix compiler errors

- We expect you at this point to be proficient enough at Go to fix most compile errors on your own.
- If there is a type you are not familiar with, first search the starter code for that type (grep -rn type typeName)
- If it is not in the starter code, search Google.
- If it is not on Google, then ask on Piazza/Office Hours.



Step2: Fix segfaults (panics)

- The only common ways you will get a segfault in Golang are by dereferencing a nil pointer or accessing an out-of-bounds index in an array.
- If you do get a segfault, Go will provide you with a stack trace with line numbers.
- If you find a segfault, you are expected to **fix it on your own** (or at least put a reasonable amount of effort). When I say fix, I mean fix, not mask.
- The exception is segfaults that occur in the starter code or Go libraries.

Step3: Hand simulate the system behavior

- Sample Scenario 1:
- Running the TestXXX test and I got a time out error.
  - Potential bugs: 1. Spinlock 2. Deadlock 3. Block at a channel 4.Did not close properly …
- Understand the corresponding functions for the test case. The tests are generally readable. ("grep <testname> *" in the lsp folder to figure out which file contains a given test.)
- Use logging statement to track the state.
- Check all the go routines (where you are running infinite loops) and reason through all the cases you have. See if you are blocked somewhere.

Step3: Hand simulate the system behavior

- > *Check all the go routines (where you are running infinite loops) and reason through all the cases you have. See if you are blocked somewhere.*
  - It may be helpful to get a printout of all current goroutines and their running/blocked status.
  - More ways to do this: https://stackoverflow.com/questions/19094099/how-to-dump-goroutine-stacktraces

Step3: Hand simulate the system behavior

- Sample Scenario 2:
- Running the srunner & crunner_sols but the srunner hangs (considering checkpoint implementation).
- Check the srunner/crunner.go file, the reference binary files also use the same format to run the reference solution.
- Reason through your code and check:
  - The connection is established correctly.
  - My server can read DATA from client.
  - My server sends ACK back.
  - My server write DATA to client.
  - …
- A good exercise: walk through a hand simulation with your partner!
  - Or, explain it to a pet, rubber duck, etc.
  - Often you will realize while explaining a piece of code that it is not doing what you thought it was.

Step4: Fix race condition

- When you run your tests with go test –race, Go will tell you if it detects any data races. This will include line numbers for the offending data accesses.
- You are expected to be able to fix any races that are detected by the race detector. If there is a race detected, it‘s time to bring out the Banker example from the first recitation.
- Change the parameters of your test cases, and run them multiple times to tease out race conditions. Even if it only detects a race 1 out of 10 runs, you should still fix that race. (Or Part B will be miserable...)

Step5: Isolate the problem using logs/prints

- Starting next week, we no longer accept Piazza questions or Office Hours questions about an end-to-end failure accompanied by the entirety of your source code.
  - This is not because we don't want to help you – we do! But looking at your code and seeing what is wrong in 10 minutes is very hard, so the best we can do is just guide you through these 5 steps. It is more efficient if you do the steps yourself and then ask us targeted questions.
- Questions about your overall system design are still welcome.
  - Questions like “how should my server handle … case” is something you should consider and design, but do not ask on Piazza please (to avoid giving away an answer to everyone).
- We expect you to isolate the exact goroutine/lines of code that are behaving differently than you expected before asking for help.

Step5: Isolate the problem using logs/prints

- Some ideas:
  - If you are having client/server coordination issues, print out every packet sent/received.
    - Consider defining a wrapper function around Read/Write funcs that prints the packet, then use that instead of Read/Write directly.
  - Same goes for channels
  - Don't be afraid to write your own tests, or call our general test functions with different arguments! Try to find a minimum broken example.
    - E.g. if you have a bug that occurs in one of our multiple-client tests, first test having multiple clients connect but do nothing else; then have one of them send a message after connecting; then have multiple of them send messages after connecting; then mix in reordering, etc. until the bug appears.

5 steps before asking for help

1. Fix compile errors
2. Fix segfaults
3. Hand-simulate system behavior
4. Fix race conditions
5. Isolate the problem using prints/logs

Ask precise, concise questions:

e.g: I noticed that in the SlowClient tests, the clients are getting responses for the wrong keys. This is what happens in my system when it receives a Get request... (explain in pseudocode)

My code passes locally but not on Gradescope

- Many students have had this issue
- We will post on Piazza detailing common performance issues. (Stealing from a previous TA)
- One additional thing you can do: use top/htop
- If your program is pinning all your CPU cores at 100% when running, that indicates a performance problem. At least for P1, a good implementation will not use more than around 20% CPU on a modern machine for any of our tests.

Words of advice

- Always understand *why* your code works.
- Don’t be afraid to rewrite part or all of your code.
- Sometimes you can catch subtle mistakes by writing it a second time – This is how you learn in general.
- You do have a partner to work together!

Further Reading

- David Andersen, "Software Engineering for Systems Hackers", https://www.cs.cmu.edu/~dga/systems-se.pdf
  - Linked on course syllabus webpage
  - Chapters 5.1 & 11