Advanced - Debugging

EECS 201 Winter 2026

Submission Instructions

This assignment is an "online assignment" on Gradescope, where you will attach your files and answer some questions.

Checkoff

In addition to submitting this assignment, you will need to attend a lab session after the deadline to get the assignment graded. You have a window from the day after to deadline until the end of the following week. For instance, if this assignment is due on a Tuesday, you have from that Wednesday until the following Friday (10 days later) to attend lab and get checked off. If the assignment is due on a Thursday, you still have until the following Friday, resulting in a shorter window for that assignment (8 days in this case).

To earn credit, be prepared to do the following:

- Explain your approach for the assignment. What documentation did you read, and how did you find it? What did you try that didn't work, and what was your process for figuring out how to fix it?
- Summarize what you learned from the assignment.
- Discuss how you plan on implementing the tools in this assignment and what you learned in your future projects.

I suggest keeping these in mind while doing the assignment, and writing down your thoughts so that you can review them before the checkoff.

Preface

In this assignment you'll be provided yet another zipped archive containing some starter files.

`https://www.eecs.umich.edu/courses/eecs201/wn2026/files/assignments/adv-debug.tar.gz`

1 Attaching to a running process (5)

This part may or may not work on macOS: try it on the course server. Occasionally you may want to debug an already running process with GDB or LLDB. Perhaps it's some sort of daemon process that you want to catch in the act or maybe it's some interesting setup that you have concocted *cough cough*. Regardless, GDB and LLDB have the ability to attach to already running processes when provided with the process ID ("PID") of a process you want to debug (and in the case of LLDB, it can also even attach to a process via the program's name!).

In `adv-debug/attach` there is a program called `revd` that will reverse strings provided to it and log them. It is intended to run in the background, using a special type of file called a *named pipe* or *FIFO* (like a pipe used to string commands together, but accessible as a file) as input. It uses the FIFO as a form of inter-process communication (IPC): a process that wants to communicate will write to the FIFO file and `revd` will read from it.

The Makefile has targets for building and running the application. `test-producer.sh` is a sample script that communicates with `revd`, forever providing it with a random word every second (until you `^C`). Try running the `run` target and then run `test-producer.sh`. In another terminal run `$ tail -f revd.log` to continuously print out updates to the log file. You should see that the `revd` process is chugging along (for extra fun, run some more instances of `test-producer.sh` or write another program/script that produces more input for `revd`).

Now onto the debugging part: the goal of this exercise is to *attach* GDB or LLDB to this running process. There's multiple ways of going about this: I'll leave it up to you to review the GDB/LLDB manual or manpage or other random

resources on the internet.

Notes:

- **If you are using WSL, the Windows filesystem (i.e. stuff under /mnt/c) does not support FIFOs. This can only be done on Unix filesystem side (i.e. everything not /mnt/c).**
- **Don't do this bullet on the course server: you don't have permissions to, and it's already set for you.** Newer versions of the Linux kernel now by default have security measures in place that prevent the tracing of processes, which GDB does to attach to a process. If you are unable to attach to the process and there is some warning/message about `ptrace` not being permitted, or some other permission denied,

  run `$ sudo sh -c "echo 0 > /proc/sys/kernel/yama/ptrace_scope"`. This will invoke the `sh` shell under `root` and write a 0 into the `/proc/sys/kernel/yama/ptrace_scope` file which serves as the way to configure the security level for tracing. **Don't do this on the course server! You don't have the permissions to modify system-level settings on my server ;p**
- `make run` will run `revd`.
- `make kill` will kill `revd` (since it has been backgrounded it's a tad more involved to stop it).
- If you're doing this on the course server, please be nice and kill your `revd` when you're done. The server will kill running `revd` processes that are older than 5 minutes to free up resources, so don't be alarmed if you find them terminated.
- `pgrep`, `pidof`, and `ps a` can help you find the PID of `revd`.
- If you're on the course server, `ps u` might be of greater use.

☐ What command-line commands did you run in the process of attaching GDB or LLDB to the running `revd` process? Be specific!

☐ If you look at the `revd.log` file you'll notice that it isn't reversing the string properly. Fix this issue and submit the fixed version. (By the way, string reversal is a pretty common algorithmic interview question)

2 How do you debug? (5)

Write a paragraph or two about what your process is when you go about debugging your programs. What do you look for? Do you start off with print statements before heading to your debugger? Let me know about your process for it!