CS 162 HW 0

GDB basics

Now we're going to use a sample program, `map`, for some GDB practice. The `map` program is designed to print out its own executing structure. Before you start, be sure to take a look at `map.c` and `recurse.c` which form the program. Once you feel familiar with the program, you can compile it by running `make map`.

Important: To run the program please use `i386-exec ./map`.

If you're curious: The "i386-" tools in this homework are necessary to ensure consistent results across architectures. "i386" was the first 32-bit x86 CPU, which is the architecture we'll be using in the Pintos projects. We've installed wrapper scripts around `qemu-i386` to enable x86 user space emulation on your machine—even if it's ARM-based. This allows you to run x86 executables on any architecture by "imitating" a virtual CPU. If you're (even more) curious about how this magic works, take a look at QEMU!

Write down the commands you use to complete each step of the following walk-through. Be sure to also **record and submit your answers to all questions in bold to Gradescope**. We highly recommend this site for an easy-to-read GDB refresher.

1. Run GDB on the `map` executable by running: `bash i386-gdb-map.sh`. This ensures a consistent environment. Do **NOT** use `gdb map` or `i386-gdb map` directly; these will lead to different results.

2. Set a breakpoint at the beginning of the program's execution.

3. Continue the program until the breakpoint.

4. **What memory address does `argv` store?**

5. **Describe what's located at that memory address. (What does `argv` point to?)**

6. Step until you reach the first call to `recur`.

7. **What is the memory address of the `recur` function?**

8. Step into the first call to `recur`.

9. Step until you reach the `if` statement.

10. Switch into assembly view.

11. Step over instructions until you reach the `call` instruction.

12. **What values are in all the registers?**

13. Step into the `call` instruction.

14. Switch back to C code mode.

15. Now print out the current call stack. *Hint: what does the `backtrace` command do?*

16. Now set a breakpoint on the `recur` function which is only triggered when the argument is 0.

17. Continue until the breakpoint is hit.

18. Print the call stack now.

19. **Now go up the call stack until you reach `main`. What is the return address to the `main` function?**

20. Now step until the return statement in `recur`.

21. Switch back into the assembly view.

22. **Which instructions correspond to the `return 0` in C?**

23. Now switch back to the source layout.

24. Finish the remaining 3 function calls.

25. Run the program to completion.

26. Quit GDB.
