ECE 391 Debugging Tutorial

Presented by HKN

Download the Code:

- Execute the following command in a terminal on your machine:
- *git clone git://github.com/HKNTutorials/391-Tutorial.git*
- This will create a folder named 391-tutorial and download the code there.

printargs: A GDB Example

- To execute the command *<program> <args>* in GDB, run the two commands
  - *gdb <program>*
  - *run <args>*
- Try this on the program “printargs”.

Useful Commands

- *break <file>:<linenumber> (or "break function_name")*
  - Set a breakpoint at a particular place in the code
- *print <expression>*
  - Evaluate the expression (using C syntax) and print the result once.

Other Useful Commands

- *display /i $pc*
  - Displays the instruction in the processor at the moment.
- *display /<n>w $esp*
  - Shows the first n 32-bit values, starting with the memory address given by esp: the top n words on the stack.
  - Use “x” instead of “print” to display things once here.

Other Useful Commands

- *info regs*
  - Displays the state of many commonly-used processor registers.
- *info locals*
  - Lists local variables.
- *Backtrace* or *bt*
  - Displays all the functions that have been called at the current point of execution.
  - Useful for debugging segfaults.

MP3 Protip: Command Files

- GDB can accept text files containing GDB commands as an input option. All commands in the file are executed when GDB loads.
- This is useful when you want to automatically have certain breakpoints set or variables displayed.
- Syntax: *gdb -x <file>*
- Try setting a few breakpoints in printargs automatically using gdb.

Exercise 1: protocol

- This is an implementation of run-length encoding.
- This code is buggy; some of the test cases fail. Can you figure out why?

Assembly Debugging Tips

- use nexti / stepi to move 1 instruction at a time
- you can refer to registers within gdb. I.e. "print $eax" prints the value of eax.
- GDB interprets everything like it is C. E.g. "print *$eax" would treat eax as a pointer and access the associated memory; normal assembly syntax is not understood.
- the disassemble command will disassemble the current function
- use "display/i $pc" to display the next instruction

Exercise 2: assembly_example

- Several odd things go on in this code. While debugging it, you should try to understand how to step through assembly & inline assembly. Fortunately, it's short, so stepping through the whole thing is feasible.
- Hint: there are exactly 2 bugs in this code.

More Exercises

- fibonacci: This one is fairly straightforward.
- nondeterministic - sometimes this crashes, sometimes it doesn't. See if you can figure out how to reliably reproduce the bug - that is, figure out an input that makes it crash (hint: what are all the inputs to the program?)
- reverse_list: this does some linked-list manipulation, but is buggy.

And a few more Exercises

- arewethereyet: This program uses signals, which are essentially a userspace software version of interrupts. Why does it crash?
- string_modification: The C code in this program is difficult to understand… so, try turning it into assembly and finding the problem!

Useful Command Summary

- *display /i $pc*
- *display /<n>w $esp*
- *info regs*
- *info locals*
- *bt*
- disassemble
- Other commands are included on the 391 GDB handout.