Debugging Practice

Devin J. Pohly <djpohly@cse.psu.edu>

Today

- Debugging workflows
  - How do I debug when X happens?
- Practice exercises

Reminders

- If you want to debug a program, you need to recompile it with the `-g` compiler flag (in CFLAGS)
- Run `gdb progname` to start GDB
- If the program takes arguments or needs I/O redirection, that comes later when you give GDB the `run` command, e.g.:
  - `run arg1 arg2 < in.txt`

General idea

- When debugging, you are trying to deduce several things:
  1. What happened?
  2. Where did it happen?
  3. Why did it happen?
     - What is the immediate cause?
  4. How did we get to this point?
     - What is the root cause?

Segfaults

- "Received signal SIGSEGV"
- What happened?
  - The program tried to access memory illegally
  - Address not mapped, or permissions/protection flags disallow the access
- Clues
  - Pointers! Variables are always stored in valid memory, but what they point to may not be.
  - Often a loop that increments a pointer or array index

Debugging segfaults

- Where did it happen?
  - Run the program and let it segfault
  - `bt` or `where` command
    - If this doesn't look right, the stack is probably smashed. See "stack smashing" instead.
  - Note: `info proc map` to show current memory mappings
- Why did it happen?
  - Figure out which pointer or array is being accessed that is not in valid memory
  - `info locals`, `info args`, or `print` the value of variables

Failed assertion

- Assertion message and "Received signal SIGABRT"
- What happened?
  - Programmer stated an assumption, and it wasn't true for some reason
    - Often pre/postconditions
  - GDB puts you right at the assertion
- Clues
  - The condition specified in the assertion
  - Any comments explaining it

Debugging assertions

PENNSTATE 1855

- Where did it happen?
  - Run the program and let the assertion fail
  - `bt`/`where`
  - `list` to see the surrounding lines
- Why did it happen?
  - Assertion condition will tell you
  - Check the values of the variables that are involved using `print`.
  - Is the assertion correct? Sometimes the programmer overlooks a case when it can legitimately be false.

Stack smashing

- Stack smashing warning or segfault
- What happened?
  - Overwrote important data on the stack
    - Return address
    - Variable values
    - Pointer values
  - Usually a buffer overflow
- Clues
  - Using pointers or array buffers
  - Using dangerous functions like `strcpy`, `strcat`, `gets`, `sprintf`
  - Incrementing pointer or array index in a loop

Debugging stack smashing

- Where did it happen?
  - Tricky because the stack is corrupted
  - Usually in the current function, but we still don't know the call history
- Why did it happen?
  - There is probably a buffer variable or pointer in this function; try `info args` to see what it might be.
  - Check to see if an array index is too high, or if there is a pointer that points somewhere within the stack segment (see `info proc map`).

Memory corruption

- Memory corruption or double free message
  - Or a segfault
- What happened?
  - Problems with dynamic memory allocation on the heap
  - Unmatched `malloc` and `free`
  - Calling `free` twice
  - Buffer overflow in dynamically allocated memory

Debugging corruption

- This is difficult to do with a debugger alone
  - In simple cases you can set a breakpoint on `malloc` and `free` to see what is happening
  - But this won't show you some things, like a heap buffer overflow
  - Other tools such as Valgrind exist to help with this

Logic errors

- Program has unexpected behavior
  - Most common, but hardest to find!
  - Programmer assumed something but didn't assert it
  - Or just a mistake, it happens

Debugging logic errors

PENNSTATE 1855

- Where did it happen?
  - Since there isn't a crash or fatal signal, you need to set some breakpoints before issuing the `run` command.
  - For an infinite loop, you can hit Ctrl-C to break to GDB
  - Often best to break somewhere before the important code, check to see if everything is OK, then step through until something looks wrong.
- What happened?
  - Depends entirely on what the problem is.
  - Use your intuition to find the programming mistake.

The last step

- How did we get to this point?
  - Figure out which variables are "interesting" based on the specific error
  - Use `up` and `down` to navigate stack frames, then examine variables in the functions that called this one using the same commands as above.
  - Set a breakpoint earlier in the program using `break`, watch variables using "`display` *`varname`*", and step through with `step`/`next` to watch what happens.

Practice time

```
$ wget -U mozilla tiny.cc/311debug
$ tar -xvzf 311debug
$ cd debug
$ make
$ ./fixme
$ gdb fixme
```

…

Commands/abbreviations

- run
- continue
- break
- delete
- next
- step
- finish
- quit
- help
- backtrace/where
- up
- down
- frame
- print
- x
- info args
- info locals
- info proc map