# GDB

GBU Debugger

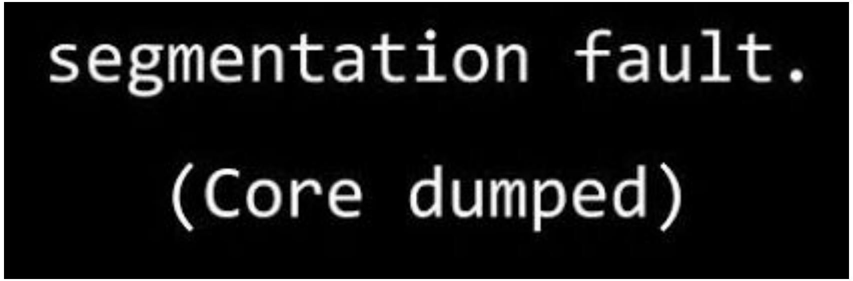
# Why use GDB? [and not only print statements)

- **Examining call stack** - seeing how functions are called and what arguments are passed in
- **Ability to pause program** at any point and see complete state
- **No code modification** reduces clutter and risk of adding errors
  - **Time efficiency** in finding where code deviates from your own mental model

# When to Use GDB

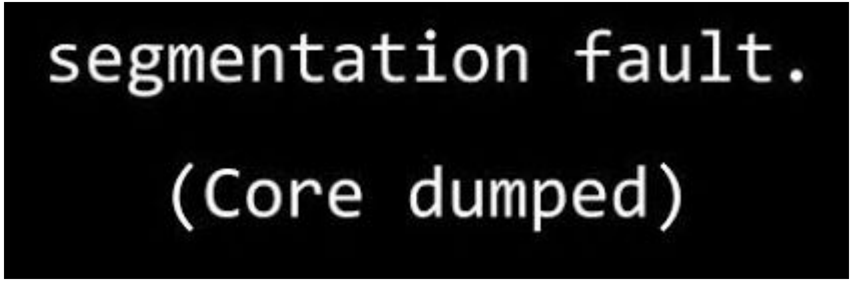
- When a print statements leads to more questions than answers…

# When to Use GDB

- When a print statements leads to more questions than answers…
- 

```
segmentation fault.
(Core dumped)
```

# When to Use GDB

- When a print statements leads to more questions than answers…
- 

  ```
  segmentation fault.
  (Core dumped)
  ```

  Debuggers allow you to inspect the state of the program right before this happens - and figure out what went wrong

# Getting Started

- In one terminal run make qemu-gdb
- In a second terminal:
    - Run gdb
        - **riscv64-unknown-elf-gdb** on Mac or Athena
        - **gdb-multiarch** on Debian, Ubuntu or WSL
        - **riscv64-linux-gnu-gdb** on Arch Linux

# GDB Commands

Run **help <command-name>** if you’re not sure how to use a command

All commands may be abbreviated if unambiguous: c = co = cont = continue

Some additional abbreviations are defined, eg. s = step and si = stepi

# Breakpoints

**break <location>** sets a breakpoint at the specified location

Locations can be memory addresses, or names

Modify breakpoints using **delete, disable, enable**

**info breakpoints** prints information about breakpoints

# Running

Continue runs code until a breakpoint is encountered or you interrupt it with control-c

Finish runs code until the current function returns

**advance <location>** runs code until the instruction pointer gets to the specified location

# Layouts

GDB has a text user interface that shows useful information like code listing, disassembly, and register contents

**layout src** shows you the source file

**layout asm** shows the assembly

**layout split** shows both the assembly and the source

# Stepping

**step** runs one line of code at a time, When there is a function call, it steps into the called function

Next does the same thing, except that it steps over function calls

**stepi** and **nexti** do the same thing for assembly instructions rather than lines of code

All take a numerical argument to specify repetition (e.g. step 5). Pressing the enter key repeats the previous command

# Examining

**x** prints the raw contents of memory in whatever format you specify (**x/x** for hexadecimal, **x/i** for assembly, etc).

**print** evaluates a C expression and prints the results as its proper type. It is often more useful than x.

The output from p *((struct elfhdr *0x1000) is much nicer than the output from x/13x 0x10000

# More examining

**`info registers`** prints the value of every register

**`info frame`** prints the current stack frame

**`info locals`** prints the value of every local variable

**`backtrace`** prints the backtrace of all stack frames

- **`frame`** lets you jump between frames

# Quality-Of-Life [breakpoints]

**`tbreak`** creates a temporary breakpoint - which is automatically removed after hitting it once.

**`advance`** will attempt to run the program to the location specified - and will automatically stop execution if it never gets there

**`command`** allows you to create a mini-script to run as soon as a breakpoint is reached

# Quality-Of-Life [breakpoints]

- **`break <location> if <condition>`** sets a breakpoint at the specified location, but only breaks if the condition is satisfied.

- **`cond <number> <condition>`** adds a condition on an existing breakpoint.

# Quality-Of-Life [stepping]

**`skip`** will automatically skip over any function when stepping through for the remainder of the execution of the program

- **`info skip`** shows current skip directives
- **`skip enable`** and **`skip disable`** enable/disable skip directives

# Further Reading

GDB is *extremely* powerful - and worth your time to spend more time learning.

We've barely scratched the surface: some other interesting things to note are

- **`watchpoint`** : stop execution only when a certain variable is changed
- **`symbol-file`** :debug files outside of kernel (like in user)

There are many "GDB quick starts" online! Check them out!

# Addendum

Checkoffs for Lab 2 should have been emailed to the designated students.

Please check your email!