Week 3.b
CS6640
01/23 2026
https://naizhengtan.github.io/26spring/

- [x] 0. admin
- [x] 1. normal vs. kernel debugging
- [x] 2. Memory layout in egos
- [ ] 3. gdb

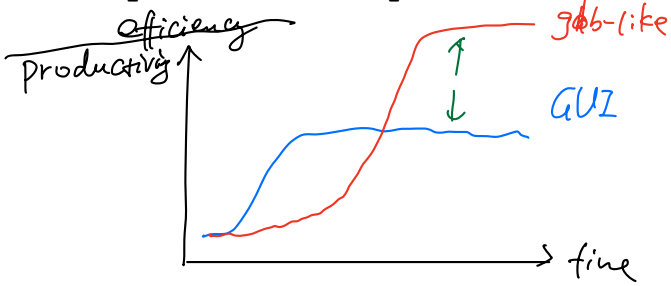
---

typos

Q: When you debug, what usually takes more time?
(e.g., finding where the bug is, or understanding why it happens?)

Q: What is your typical approach to debugging a program?

Q: What is one debugging habit you think helps the most?

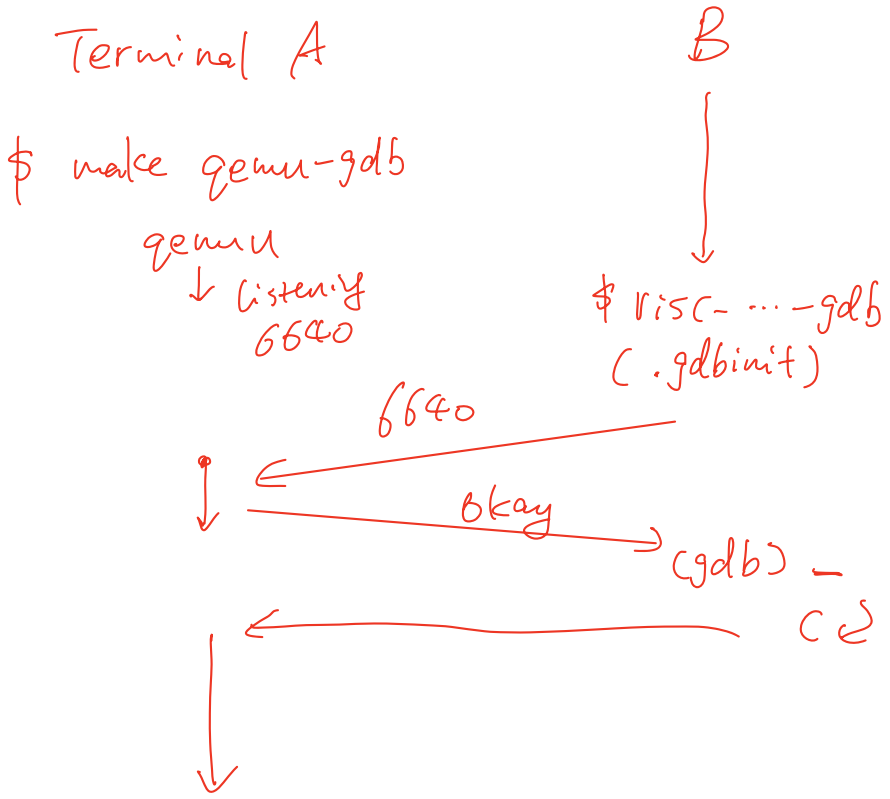


* "normal" c program
  + you do not need to understand hardware details (like CPU)
  + you have clear error messages
  + you do not have to worry about touching important memory
    (the program will be killed)
  + you do not use addresses directly
  + you have a nice address space containing your program only
  + you have a lot of tools (like IDE)

* kernel programming
  - you need to understand hardware details (like CPU)
  - you have semi-clear error messages (if you know CPU)
  - your have to worry about touching important memory
    (the kernel will write something to there and later crash)
  - you sometimes need to use addresses directly
  - you do not have a nice address space
  - you have limited yet powerful tools

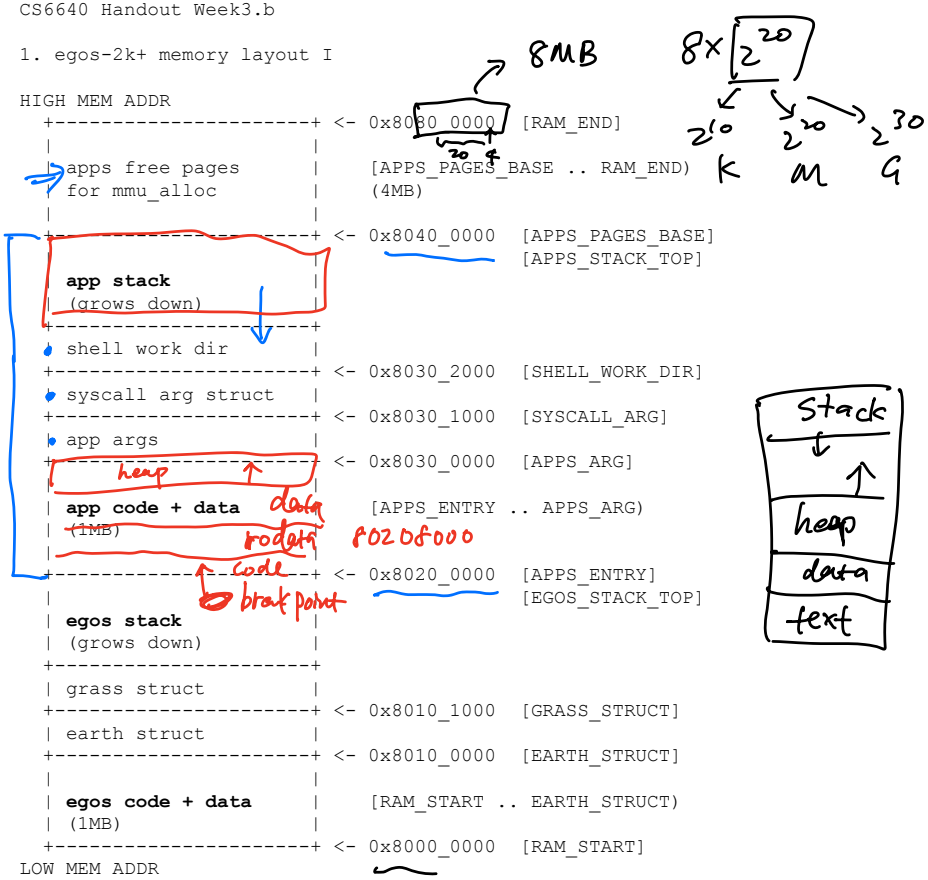
* ULT labs? a mix of the two
  + you need to understand hardware details (like CPU)
  - you have semi-clear error messages (if you know CPU)
  - your have to worry about touching important memory
  + you do not use addresses directly
  - you do not have a nice address space
  - you have limited yet powerful tools

* some debugging principles
  - "die" earlier than later
  - use "assert" a lot
  - use "printf" but don't trust "printf" entirely
  - binary-printf is still useful
  - use "static analysis" more often (e.g., "git diff")



CS6640 Handout Week3.b

1. egos-2k+ memory layout I



2. gdb cheat sheet

```
Multi-core support
(gdb) info threads          list all threads known to gdb
(gdb) thread <n>            switch the current context to thread <n>

Breakpoints & watchpoints
(gdb) break main            set a breakpoint on a function
(gdb) break basic.c:101     set breakpoint at file and line (or function)
(gdb) info breakpoints      show breakpoints
(gdb) delete 1              delete a breakpoint by number
(gdb) watch expression      set software watchpoint on variable
(gdb) info watchpoints      show current watchpoints

Running the program
(gdb) c                 continue the program
(gdb) s                 a step in C; step into functions
(gdb) si                a step in asm; step into functions
(gdb) n                 a step in C; step over functions
(gdb) ni                a step in asm; but step over functions
(gdb) CTRL-C            actually SIGINT, stop execution of current program
(gdb) finish            finish current function's execution

Stack backtrace
(gdb) bt                print stack backtrace
(gdb) info locals       print automatic variables in frame
(gdb) info registers    print registers sans floats

Browsing Data
(gdb) p expr              print expression
(gdb) p/x expr            print in hex
(gdb) p/t expr            print in binary
(gdb) p/i expr            print as instructions

(gdb) x/FMT address       low-level examine command
(gdb) x/x 0x80001000      print memory in hex
(gdb) set var = expr      assign value

(gdb) display/FMT expr    display expression result at stop
(gdb) display/i $pc       print next instruction
(gdb) undisplay           delete displays

FMT (Format letters) are:
  o(octal), x(hex), d(decimal), u(unsigned decimal),
  t(binary), f(float), a(address), i(instruction), c(char), s(string)
  and z(hex, zero padded on the left).

Load a program's symbols
(gdb) add-symbol-file <elf>    load symbol table from <elf>

Quit
(gdb) quit                quit gdb
```

[borrowed and customized from
https://gist.github.com/rkubik/b96c23bd8ed58333de37f2b8cd052c30]