Week 3.b
CS6640
09/21 2023
https://naizhengtan.github.io/23fall/

1. normal vs. kernel debugging
2. Memory layout in egos ←
3. gdb
4. a tricky bug

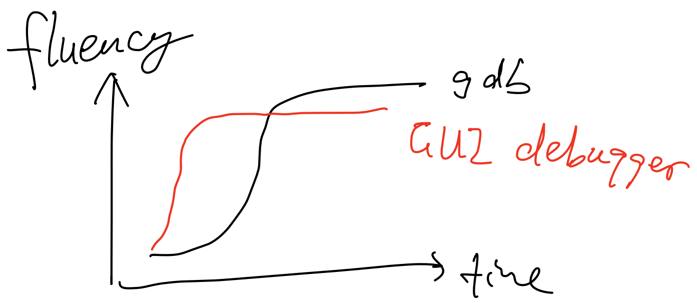
Q: How do people debug C program?

void * addr = (void*) 0xdeadbeef;

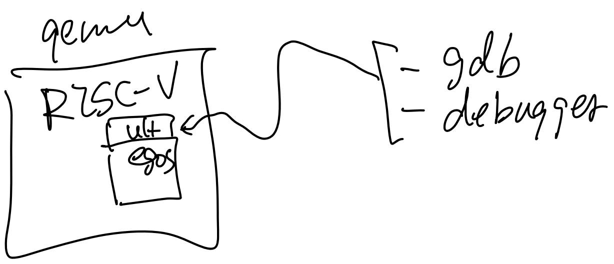
segfault

* "normal" C program
  0.5 + 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)

Q: ULT



* kernel programming
  0.5 - 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 (gdb)



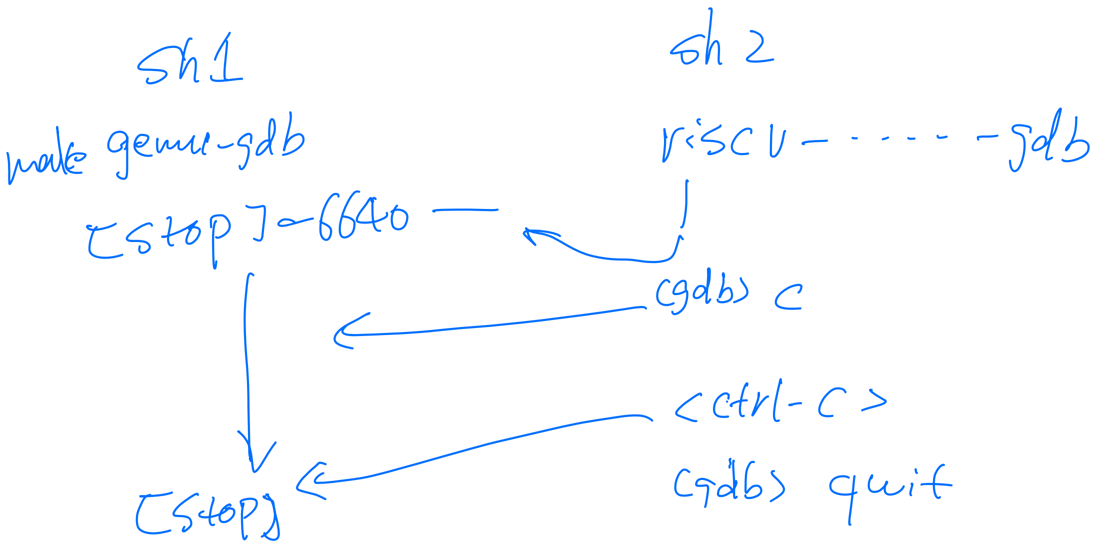
- kernel debugging principles

  - "die" eariler
  - use ASSERT more often

```
foo(int xyz){
precond → ASSERT(xyz...);
    body
post condition → ASSERT(ret..);
}
```

  - use "printf"
    but not trust it.
  - "binary-printf" is useful
    search
  - use "static analysis" more often
    ↳ git diff

- egos. design
  - earth: abstract HW
  - grass: provide services
  - apps:
    - sys apps: basic functionalities
    - user apps: hellowork, ~~ul~~t

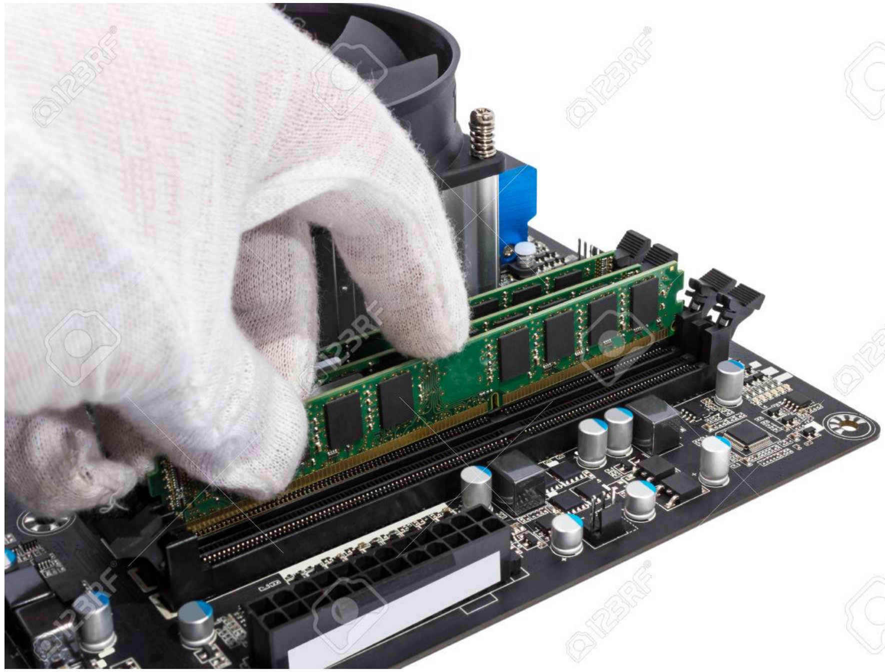


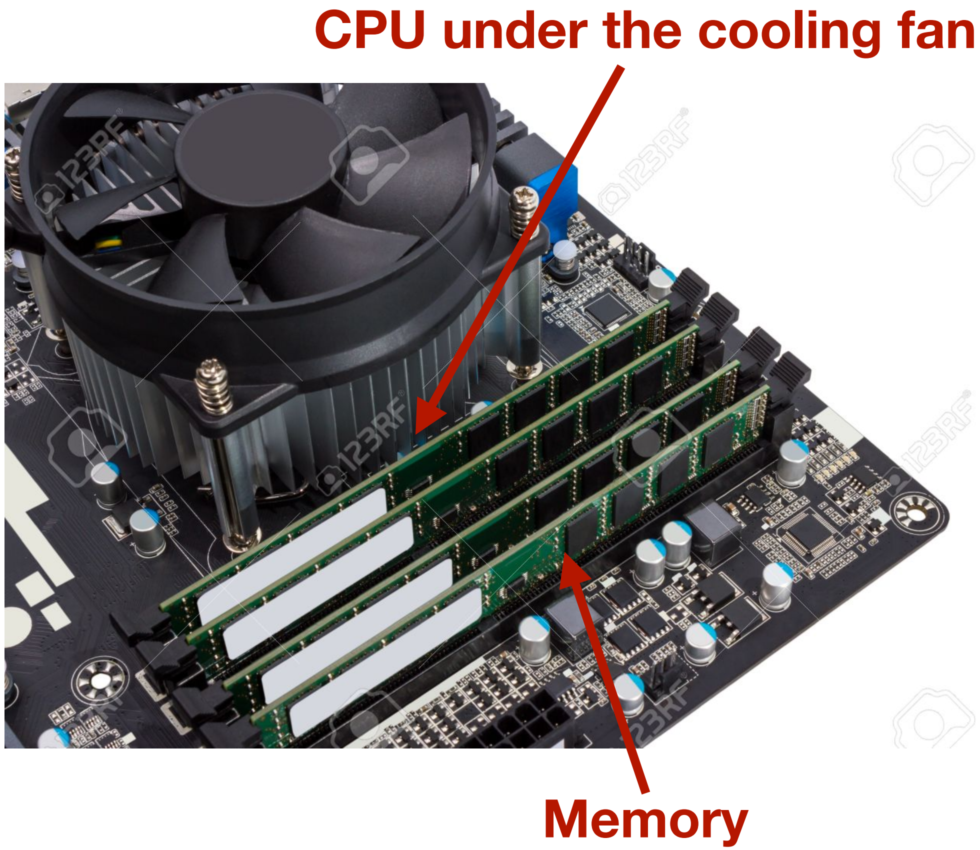
```
sh1                          sh2
make qemu-gdb                riscv-.....-gdb
[stop]-6640 ←
                             (gdb) c
                             <ctrl-c>
[stop]      ←
                             (gdb) quit
```

```
CS6640 Handout Week3.b

1. egos-2k+ memory layout

HIGH_MEM ADDR
-------  +----------------------+ <- 0x8040_0000
         |                      |    [FREE_MEM_END]
  DTIM   | free memory          |
memory   | (4MB - 16KB)         |
 (4MB)   +----------------------+ <- 0x8000_4000
         | earth interface      |    [FREE_MEM_START]
         | (128B)               |
         +----------------------+ <- 0x8000_3f80
         | earth/grass stack    |    [GRASS_STACK_TOP]
         | (~8KB)               |
         \/\/\/\/\/\/\/\/\/\/\/\/

         /\/\/\/\/\/\/\/\/\/\/\/\
         | grass interface      |
         +----------------------+ <- 0x8000_2000
         | app stack            |    [APPS_STACK_TOP]
         | (6KB)                |
         +----------------------+ <- 0x8000_0800
         | system call args     |
         | (1KB)                |
         +----------------------+ <- 0x8000_0400
         | app args             |    [SYSCALL_ARG]
         | (1KB)                |
-------  +----------------------+ <- 0x8000_0000
                                     [APPS_ARG]

         ...
-------  +----------------------+ <- 0x0a00_0000
         |                      |    [ITIM_END]
         \/\/\/\/\/\/\/\/\/\/\/\/

         /\/\/\/\/\/\/\/\/\/\/\/\
         |                      |
         +----------------------+ <- 0x0820_4000
  ITIM   | app code+data        |    [APPS_ENTRY+APPS_SIZE]
 (32MB)  | (16KB)               |
         +----------------------+ <- 0x0820_0000
         | grass code+data      |    [APPS_ENTRY]   0x8101c550
         | (1 MB)               |
         +----------------------+ <- 0x0810_0000
         | earth data           |    [GRASS_ENTRY]
         | (1 MB)               |
-------  +----------------------+ <- 0x0800_0000
LOW MEM ADDR                         [ITIM_START]
```

```
2. gdb cheat sheet

Breakpoints & watchpoints
(gdb) break main          set a breakpoint on a function
(gdb) break ult.c:10      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>

History Display
(gdb) show commands            print command history

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

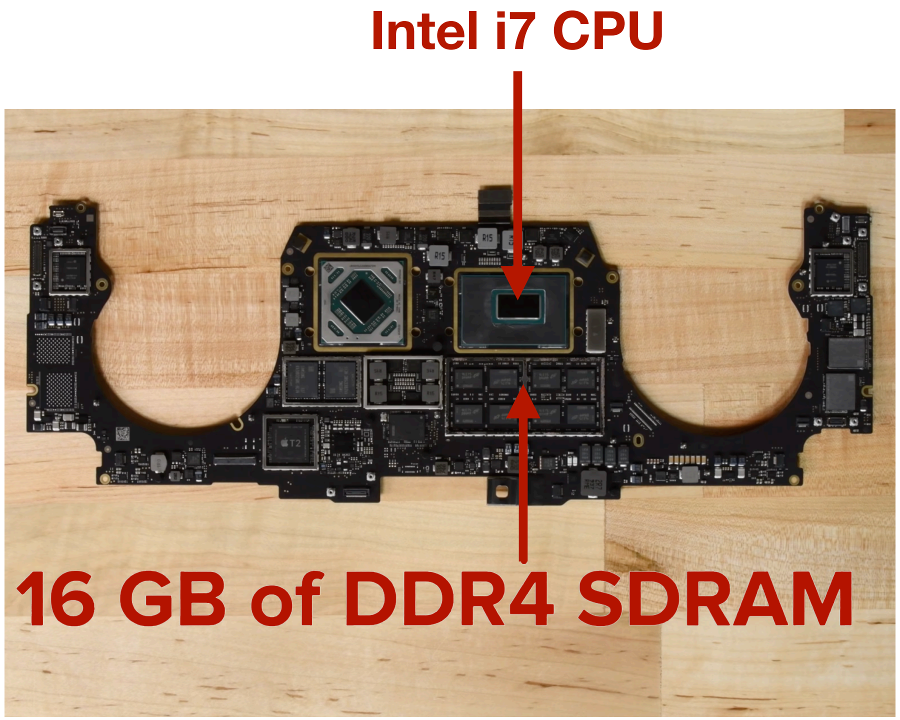
Physical memory





* Images from https://www.123rf.com/

Physical memory



* https://www.youtube.com/watch?v=Mjb12GCKycw