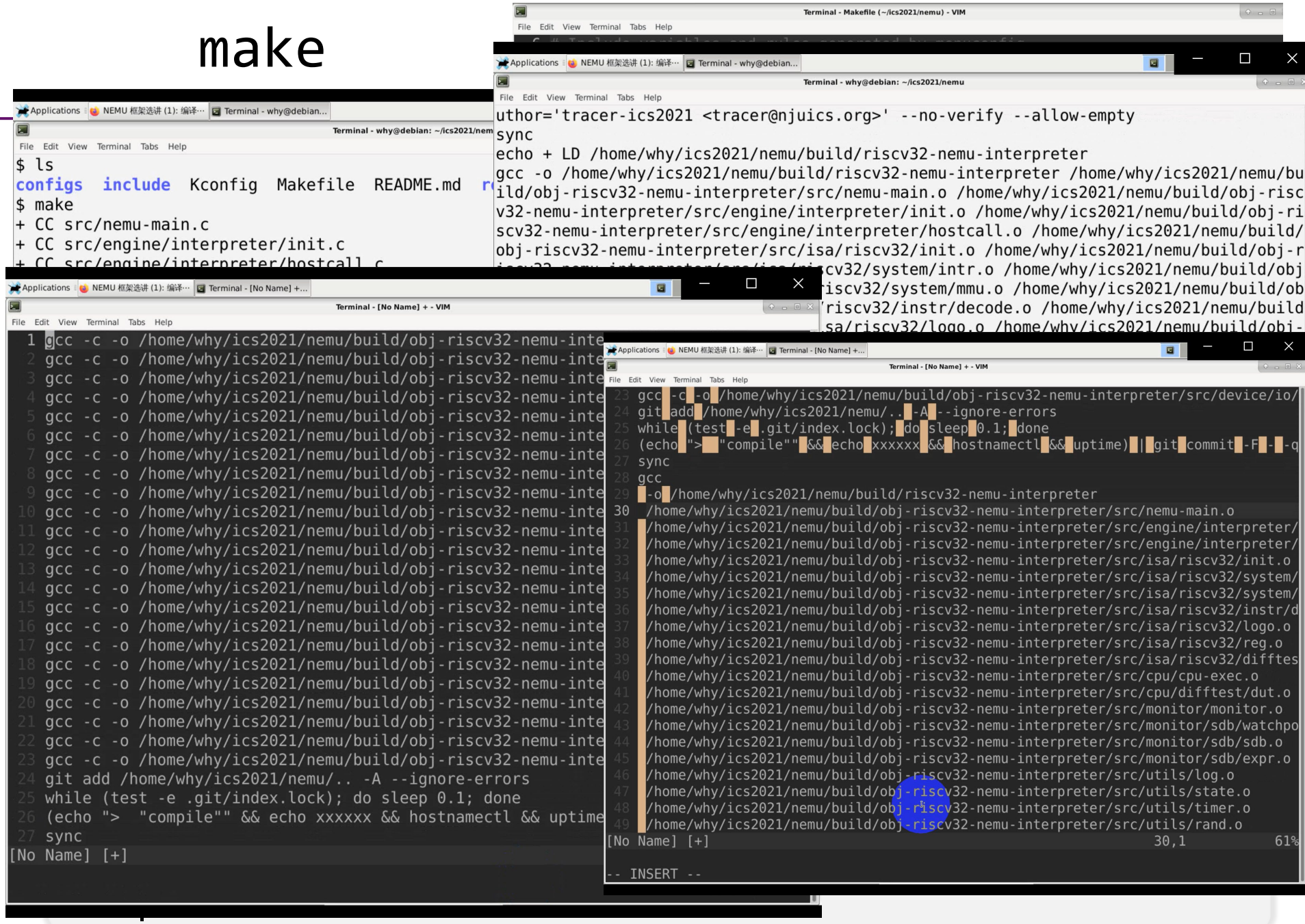
NEMU 框架选讲2: 代码导读

王慧妍
why@nju.edu.cn

南京大学 计算机科学与技术系 计算机软件研究所

make


Terminal - Makefile (~/ics2021/nemu) - VIM
File Edit View Terminal Tabs Help
Applications NEMU 框架选讲 (1): 编译··· Terminal - why@debian...
Terminal - why@debian: ~/ics2021/nemu
File Edit View Terminal Tabs Help
uthor='tracer-ics2021 <tracer@njuics.org>' --no-verify --allow-empty
sync
echo + LD /home/why/ics2021/nemu/build/riscv32-nemu-interpreter
gcc -o /home/why/ics2021/nemu/build/riscv32-nemu-interpreter /home/why/ics2021/nemu/bu
ild/obj-riscv32-nemu-interpreter/src/nemu-main.o /home/why/ics2021/nemu/build/obj-risc
v32-nemu-interpreter/src/engine/interpreter/init.o /home/why/ics2021/nemu/build/obj-ri
scv32-nemu-interpreter/src/engine/interpreter/hostcall.o /home/why/ics2021/nemu/build/
obj-riscv32-nemu-interpreter/src/isa/riscv32/init.o /home/why/ics2021/nemu/build/obj-r
Terminal - why@debian: ~/ics2021/nem
File Edit View Terminal Tabs Help
$ ls
configs include Kconfig Makefile README.md
$ make
+ CC src/nemu-main.c
+ CC src/engine/interpreter/init.c
Applications NEMU 框架选讲 (1): 编译··· Terminal - [No Name] +...
Terminal - [No Name] + - VIM
File Edit View Terminal Tabs Help
gcc -c -o /home/why/ics2021/nemu/build/obj-riscv32-nemu-inte
git add /home/why/ics2021/nemu/.. -A --ignore-errors
while (test -e .git/index.lock); do sleep 0.1; done
(echo "> "compile"" && echo xxxxxx && hostnamectl && uptime
sync
[No Name] [+]
Applications NEMU 框架选讲 (1): 编译··· Terminal - [No Name] +...
Terminal - [No Name] + - VIM
File Edit View Terminal Tabs Help
gcc -c -o /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/device/io/
git add /home/why/ics2021/nemu/.. -A --ignore-errors
while (test -e .git/index.lock); do sleep 0.1; done
(echo "> "compile"" && echo xxxxxx && hostnamectl && uptime) || git commit -F - -q
sync
gcc
-o /home/why/ics2021/nemu/build/riscv32-nemu-interpreter
/home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/nemu-main.o
/home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/engine/interpreter/
/home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/engine/interpreter/
/home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/isa/riscv32/init.o
/home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/isa/riscv32/system/
/home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/isa/riscv32/system/
/home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/isa/riscv32/instr/d
/home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/isa/riscv32/logo.o
/home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/isa/riscv32/reg.o
/home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/isa/riscv32/difftes
/home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/cpu/cpu-exec.o
/home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/cpu/difftest/dut.o
/home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/monitor/monitor.o
/home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/monitor/sdb/watchpo
/home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/monitor/sdb/sdb.o
/home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/monitor/sdb/expr.o
/home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/utils/log.o
/home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/utils/state.o
/home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/utils/timer.o
/home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/utils/rand.o
[No Name] [+] 30,1 61%
-- INSERT --

提醒

PA1：

Deadline：2022年10月13日 23:59:59

Lab1：

Deadline: 2022年10月16日 23:59:59

框架代码已上线

PA2：OJ尚未打开

PA 2.1: 2022年10月23日 (此为建议的不计分 deadline)

PA 2.2: 2022年10月30日 (此为建议的不计分 deadline)

PA 2.3: 2022年11月6日 23:59:59 (以此 deadline 计按时提交奖励分)

本讲概述

就是理解了构建的过程，NEMU代码依然很难读？

- NEMU代码导读
  - 浏览源代码
  - 启动代码选讲
  - 编辑器配置

RTFSC

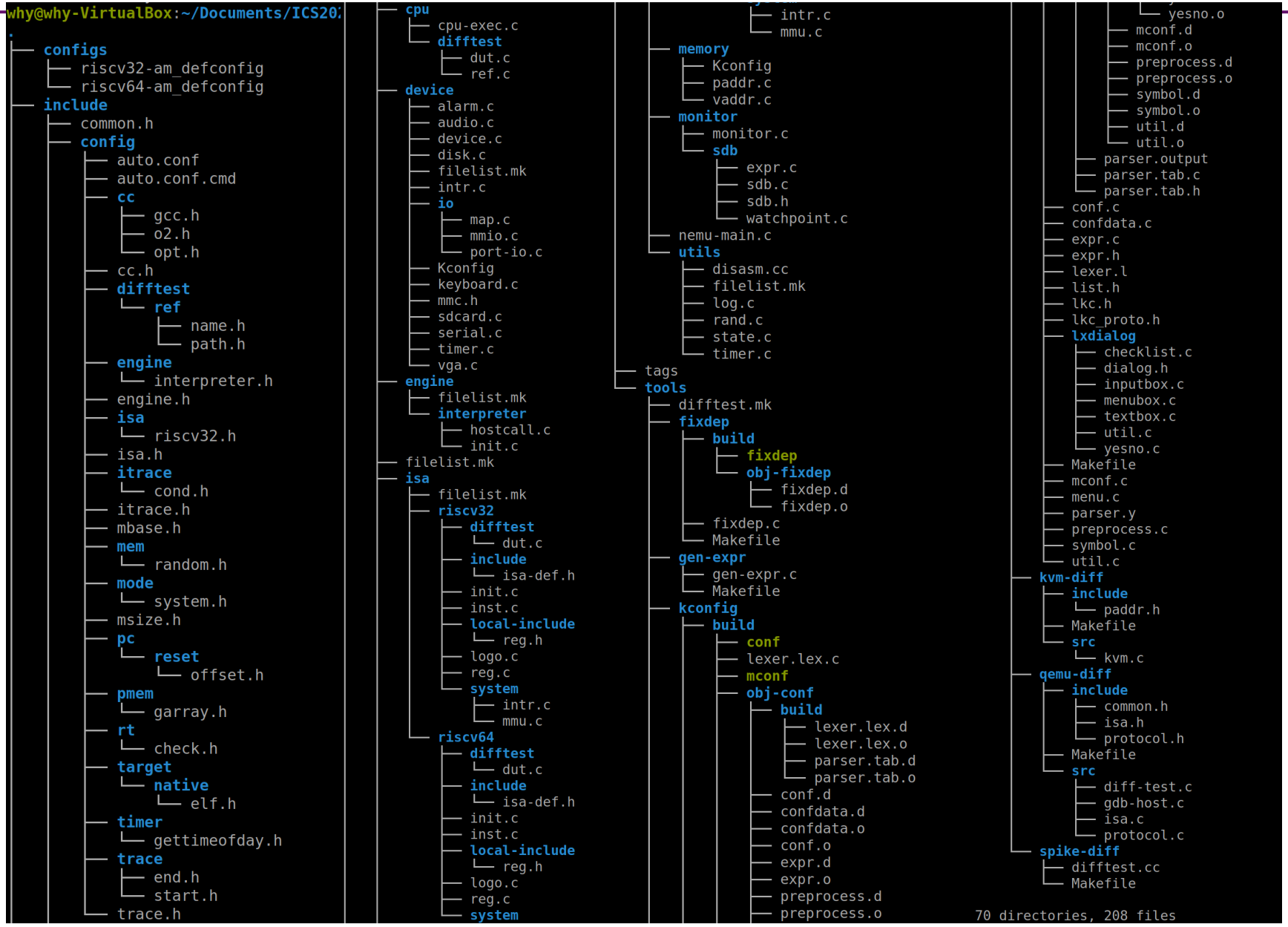
Read the friendly source code!

拿到源代码，先做什么?

NEMU对大部分同学来说是一个“前所未有大”的项目。

- 先大致了解一下
  - 项目总体组织
    - tree要翻好几个屏幕
    - `find . -name "*.c" -o -name "*.h"` (100+ 个文件)
  - 项目规模
    - `find ... | xargs cat | wc -l`
    - 5000行左右 (其实很小了)

tree


why@why-VirtualBox:~/Documents/ICS20
.
├── configs
│   ├── riscv32-am_defconfig
│   └── riscv64-am_defconfig
├── include
│   ├── common.h
│   ├── config
│   │   ├── auto.conf
│   │   ├── auto.conf.cmd
│   │   ├── cc
│   │   │   ├── gcc.h
│   │   │   ├── o2.h
│   │   │   └── opt.h
│   │   ├── cc.h
│   │   ├── difftest
│   │   │   └── ref
│   │   │   ├── name.h
│   │   │   └── path.h
│   │   ├── engine
│   │   │   └── interpreter.h
│   │   ├── engine.h
│   │   ├── isa
│   │   │   └── riscv32.h
│   │   ├── isa.h
│   │   ├── itrace
│   │   │   └── cond.h
│   │   ├── itrace.h
│   │   ├── mbase.h
│   │   ├── mem
│   │   │   └── random.h
│   │   ├── mode
│   │   │   └── system.h
│   │   ├── msize.h
│   │   ├── pc
│   │   │   └── reset
│   │   │   └── offset.h
│   │   ├── pmem
│   │   │   └── garray.h
│   │   ├── rt
│   │   │   └── check.h
│   │   ├── target
│   │   │   └── native
│   │   │   └── elf.h
│   │   ├── timer
│   │   │   └── gettimeofday.h
│   │   ├── trace
│   │   │   ├── end.h
│   │   │   └── start.h
│   │   └── trace.h
├── cpu
│   ├── cpu-exec.c
│   └── difftest
│   ├── dut.c
│   └── ref.c
├── device
│   ├── alarm.c
│   ├── audio.c
│   ├── device.c
│   ├── disk.c
│   ├── filelist.mk
│   ├── intr.c
│   ├── io
│   │   ├── map.c
│   │   ├── mmio.c
│   │   └── port-io.c
│   ├── Kconfig
│   ├── keyboard.c
│   ├── mmc.h
│   ├── sdcard.c
│   ├── serial.c
│   ├── timer.c
│   └── vga.c
├── engine
│   ├── filelist.mk
│   └── interpreter
│   ├── hostcall.c
│   └── init.c
├── filelist.mk
├── isa
│   ├── filelist.mk
│   ├── riscv32
│   │   ├── difftest
│   │   │   └── dut.c
│   │   ├── include
│   │   │   └── isa-def.h
│   │   ├── init.c
│   │   ├── inst.c
│   │   ├── local-include
│   │   │   └── reg.h
│   │   ├── logo.c
│   │   ├── reg.c
│   │   └── system
│   │   ├── intr.c
│   │   └── mmu.c
│   └── riscv64
│   ├── difftest
│   │   └── dut.c
│   ├── include
│   │   └── isa-def.h
│   ├── init.c
│   ├── inst.c
│   ├── local-include
│   │   └── reg.h
│   ├── logo.c
│   ├── reg.c
│   └── system
│   ├── intr.c
│   └── mmu.c
├── memory
│   ├── Kconfig
│   ├── paddr.c
│   └── vaddr.c
├── monitor
│   ├── monitor.c
│   └── sdb
│   ├── expr.c
│   ├── sdb.c
│   ├── sdb.h
│   └── watchpoint.c
├── nemu-main.c
├── utils
│   ├── disasm.cc
│   ├── filelist.mk
│   ├── log.c
│   ├── rand.c
│   ├── state.c
│   └── timer.c
├── tags
└── tools
    ├── difftest.mk
    ├── fixdep
    │   ├── build
    │   │   ├── fixdep
    │   │   └── obj-fixdep
    │   │   ├── fixdep.d
    │   │   └── fixdep.o
    │   ├── fixdep.c
    │   └── Makefile
    ├── gen-expr
    │   ├── gen-expr.c
    │   └── Makefile
    ├── kconfig
    │   ├── build
    │   │   ├── conf
    │   │   ├── lexer.lex.c
    │   │   ├── mconf
    │   │   ├── obj-conf
    │   │   │   ├── build
    │   │   │   │   ├── lexer.lex.d
    │   │   │   │   ├── lexer.lex.o
    │   │   │   │   ├── parser.tab.d
    │   │   │   │   └── parser.tab.o
    │   │   │   ├── conf.d
    │   │   │   ├── confdata.d
    │   │   │   ├── confdata.o
    │   │   │   ├── conf.o
    │   │   │   ├── expr.d
    │   │   │   ├── expr.o
    │   │   │   ├── preprocess.d
    │   │   │   ├── preprocess.o
    │   │   │   ├── yesno.o
    │   │   │   ├── mconf.d
    │   │   │   ├── mconf.o
    │   │   │   ├── preprocess.d
    │   │   │   ├── preprocess.o
    │   │   │   ├── symbol.d
    │   │   │   ├── symbol.o
    │   │   │   ├── util.d
    │   │   │   └── util.o
    │   │   ├── parser.output
    │   │   ├── parser.tab.c
    │   │   └── parser.tab.h
    │   ├── conf.c
    │   ├── confdata.c
    │   ├── expr.c
    │   ├── expr.h
    │   ├── lexer.l
    │   ├── list.h
    │   ├── lkc.h
    │   ├── lkc_proto.h
    │   ├── lxdialog
    │   │   ├── checklist.c
    │   │   ├── dialog.h
    │   │   ├── inputbox.c
    │   │   ├── menubox.c
    │   │   ├── textbox.c
    │   │   ├── util.c
    │   │   └── yesno.c
    │   ├── Makefile
    │   ├── mconf.c
    │   ├── menu.c
    │   ├── parser.y
    │   ├── preprocess.c
    │   ├── symbol.c
    │   └── util.c
    ├── kvm-diff
    │   ├── include
    │   │   └── paddr.h
    │   ├── Makefile
    │   └── src
    │   └── kvm.c
    ├── qemu-diff
    │   ├── include
    │   │   ├── common.h
    │   │   ├── isa.h
    │   │   └── protocol.h
    │   ├── Makefile
    │   └── src
    │   ├── diff-test.c
    │   ├── gdb-host.c
    │   ├── isa.c
    │   └── protocol.c
    └── spike-diff
        ├── difftest.cc
        └── Makefile
70 directories, 208 files

拿到源代码，先做什么?

NEMU对大部分同学来说是一个“前所未有大”的项目。

- 先大致了解一下
  - 项目总体组织
    - tree要翻好几个屏幕
    - `find . -name "*.c" -o -name "*.h"` (100+ 个文件)

find . -name "*.c" -o -name "*.h"

```
$ find . -name "*.c" -o -name "*.h"
./tools/gen-expr/gen-expr.c
./tools/fixdep/fixdep.c
./tools/kvm-diff/src/kvm.c
./tools/kvm-diff/include/paddr.h
./tools/kconfig/confdata.c
./tools/kconfig/mconf.c
./tools/kconfig/lkc.h
./tools/kconfig/symbol.c
./tools/kconfig/menu.c
./tools/kconfig/lxdialog/menubox.c
./tools/kconfig/lxdialog/dialog.h
./tools/kconfig/lxdialog/checklist.c
./tools/kconfig/lxdialog/textbox.c
./tools/kconfig/lxdialog/inputbox.c
./tools/kconfig/lxdialog/util.c
./tools/kconfig/lxdialog/yesno.c
./tools/kconfig/lkc_proto.h
./tools/kconfig/list.h
./tools/kconfig/build/lexer.lex.c
./tools/kconfig/build/parser.tab.h
./tools/kconfig/build/parser.tab.c
```

拿到源代码，先做什么？

NEMU对大部分同学来说是一个“前所未有大”的项目。

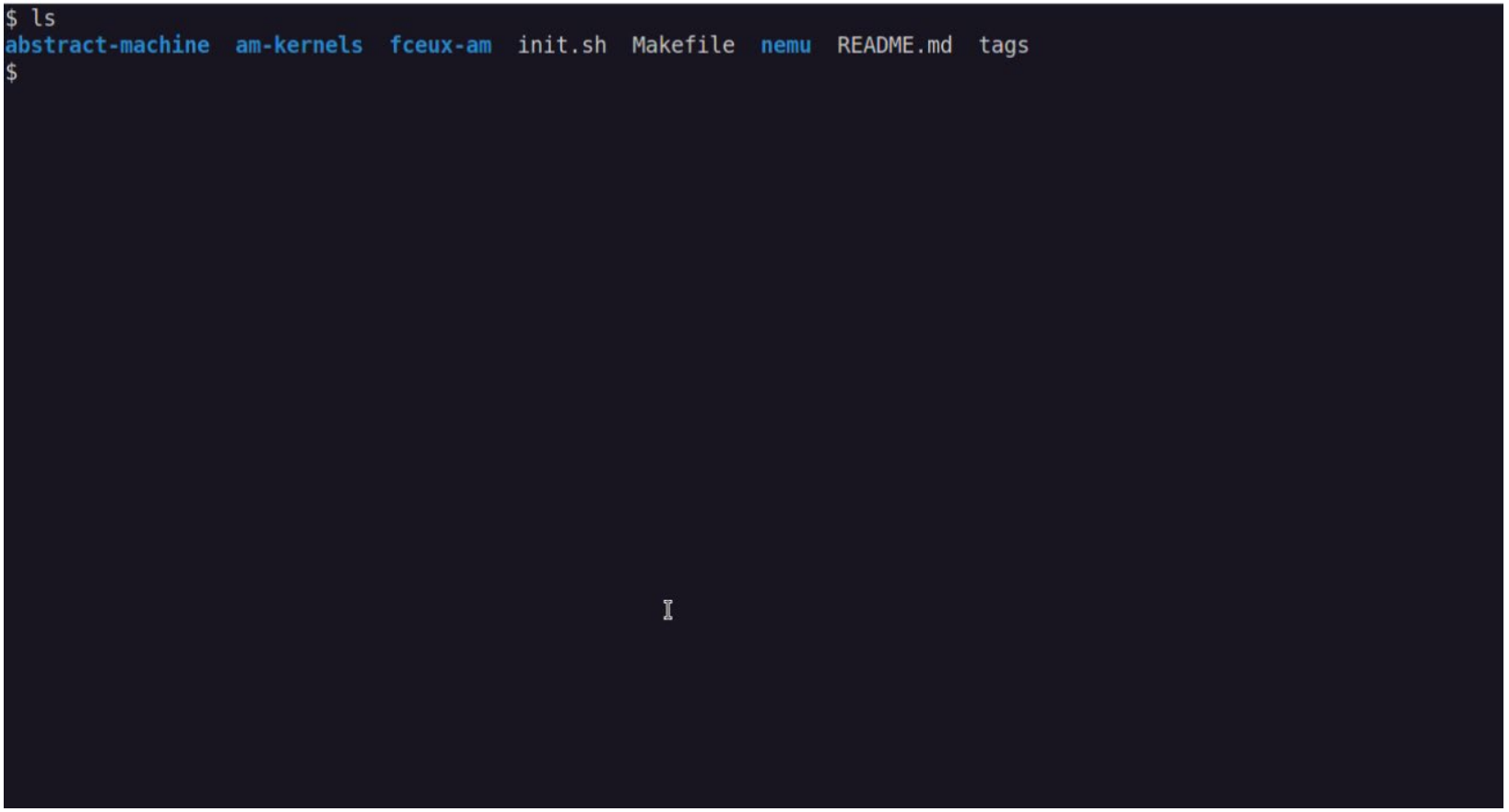
- 先大致了解一下
  - 项目总体组织
    - tree要翻好几个屏幕
    - `find . -name "*.c" -o -name "*.h"` (100+ 个文件)
  - 项目规模
    - `find ... | xargs cat | wc -l`

```
$ find . -name "*.c" -o -name "*.h" | xargs cat | wc -l
23069
```

```
$ find tools -name "*.c" -o -name ".h" | xargs cat | wc -l
17728
```

尝试阅读代码：从main开始

- C语言代码，都是从main()开始运行的。那么哪里才有main呢?
  - 浏览代码：发现main.c，估计在里面
  - 使用**IDE（vscode：Edit→Find in files)**



尝试阅读代码：从main开始

- C语言代码，都是从main()开始运行的。那么哪里才有main呢?
  - 浏览代码：发现main.c，估计在里面
  - 使用**IDE（vscode：Edit→Find in files)**

- The UNIX Way (无需启动任何程序，即可直接查看)

```
grep -n main $(find . -name "*.c") # RTFM: -n
```

grep -n main $(find . -name "*.c")

```
$ grep -n main $(find . -name "*.c")
./tools/gen-expr/gen-expr.c:13:"int main() { "
./tools/gen-expr/gen-expr.c:23:int main(int argc, char *argv[]) {
./tools/fixdep/fixdep.c:385:int main(int argc, char *argv[])
./tools/kconfig/mconf.c:1004:int main(int ac, char **av)
./tools/kconfig/symbol.c:1265:          /* for choice groups start the check with main
 choice symbol */
./tools/kconfig/menu.c:331:                     /* set the type of the remaining choic
e values */
./tools/kconfig/build/lexer.lex.c:144:/* The state buf must be large enough to hold on
e state per character in the main buffer.
./tools/kconfig/build/lexer.lex.c:2684:/** The main scanner function which does all th
e work.
./tools/kconfig/build/lexer.lex.c:4119:/* Defined in main.c */
./tools/kconfig/build/parser.tab.c:541:  "T_NOT", "$accept", "input", "mainmenu_stmt",
 "stmt_list",
./tools/kconfig/conf.c:500:int main(int ac, char **av)
./resource/sdcard/nemu.c:91:    unsigned int            blocks;         /* remaining P
IO blocks */
./src/engine/interpreter/init.c:3:void sdb_mainloop();
./src/engine/interpreter/init.c:10:  sdb_mainloop();
./src/isa/riscv64/instr/decode.c:55:def_THelper(main) {
./src/isa/riscv64/instr/decode.c:65:  int idx = table_main(s);
./src/isa/riscv32/instr/decode.c:55:def_THelper(main) {
./src/isa/riscv32/instr/decode.c:65:  int idx = table_main(s);
./src/nemu-main.c:8:int main(int argc, char *argv[]) {
./src/monitor/sdb/sdb.c:84:void sdb_mainloop() {
./src/monitor/sdb/sdb.c:97:    /* treat the remaining string as the arguments,
```

尝试阅读代码：从main开始

- C语言代码，都是从main()开始运行的。那么哪里才有main呢?
  - 浏览代码：发现main.c，估计在里面
  - 使用**IDE（vscode：Edit→Find in files)**

- The UNIX Way (无需启动任何程序，即可直接查看)

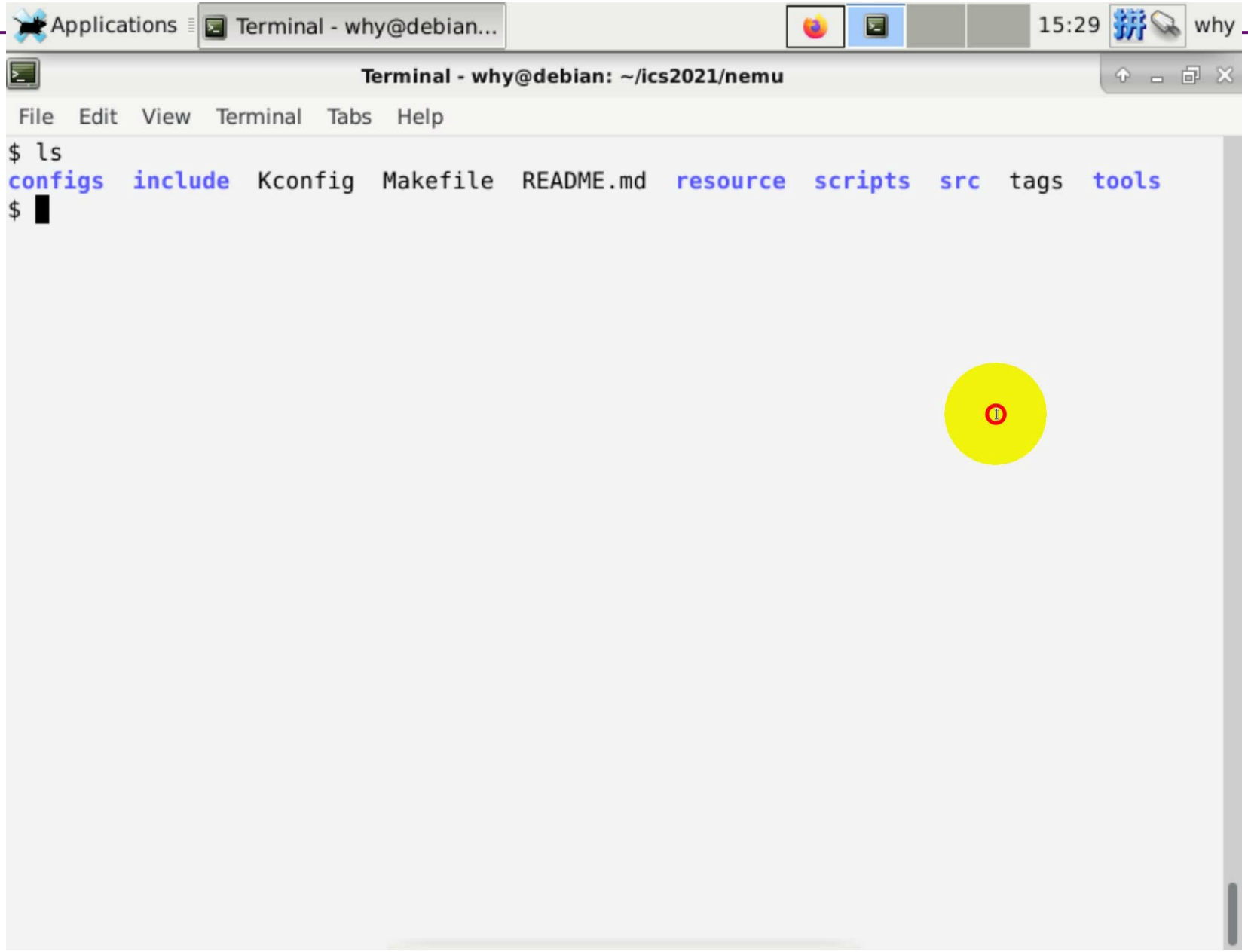
```
grep -n main $(find . -name "*.c") # RTFM: -n
```

```
find . | xargs grep --color -nse '\<main\>'
```

find . | xargs grep --color -nse '\<main\>'

```
$ find . | xargs grep --color -nse '\<main\>'
./README.md:7:The main features of NEMU include
./tools/gen-expr/gen-expr.c:13:"int main() { "
./tools/gen-expr/gen-expr.c:23:int main(int argc, char *argv[]) {
./tools/fixdep/fixdep.c:385:int main(int argc, char *argv[])
Binary file ./tools/fixdep/build/fixdep matches
Binary file ./tools/fixdep/build/obj-fixdep/fixdep.o matches
./tools/kconfig/mconf.c:1004:int main(int ac, char **av)
./tools/kconfig/symbol.c:1265:          /* for choice groups start the check with main
 choice symbol */
Binary file ./tools/kconfig/build/conf matches
Binary file ./tools/kconfig/build/obj-conf/conf.o matches
./tools/kconfig/build/lexer.lex.c:144:/* The state buf must be large enough to hold on
e state per character in the main buffer.
./tools/kconfig/build/lexer.lex.c:2684:/** The main scanner function which does all th
e work.
./tools/kconfig/build/lexer.lex.c:4119:/* Defined in main.c */
Binary file ./tools/kconfig/build/obj-mconf/mconf.o matches
Binary file ./tools/kconfig/build/mconf matches
./tools/kconfig/conf.c:500:int main(int ac, char **av)
./tools/kconfig/expr.h:70:      S_DEF_USER,             /* main user value */
./src/isa/riscv64/instr/decode.c:55:def_THelper(main) {
./src/isa/riscv32/instr/decode.c:55:def_THelper(main) {
./src/filelist.mk:1:SRCS-y += src/nemu-main.c
./src/nemu-main.c:8:int main(int argc, char *argv[]) {
Binary file ./build/riscv32-nemu-interpreter matches
./build/obj-riscv32-nemu-interpreter/src/nemu-main.d:1:cmd_/home/why/ics2021/nemu/buil
d/obj-riscv32-nemu-interpreter/src/nemu-main.o := unused
```

fzf


Applications
Terminal - why@debian...
15:29
why
Terminal - why@debian: ~/ics2021/nemu
File Edit View Terminal Tabs Help
$ ls
configs include Kconfig Makefile README.md resource scripts src tags tools
$

尝试阅读代码：从main开始

- C语言代码，都是从main()开始运行的。那么哪里才有main呢?
  - 浏览代码：发现main.c，估计在里面
  - 使用**IDE（vscode：Edit→Find in files)**

- The UNIX Way (无需启动任何程序，即可直接查看)

```
grep -n main $(find . -name "*.c") # RTFM: -n
find . | xargs grep --color -nse '\<main\>'
```

  - Fuzzy Finder
  - Vim当然也支持

```
:vimgrep /\<main\>/ **/*.c
```

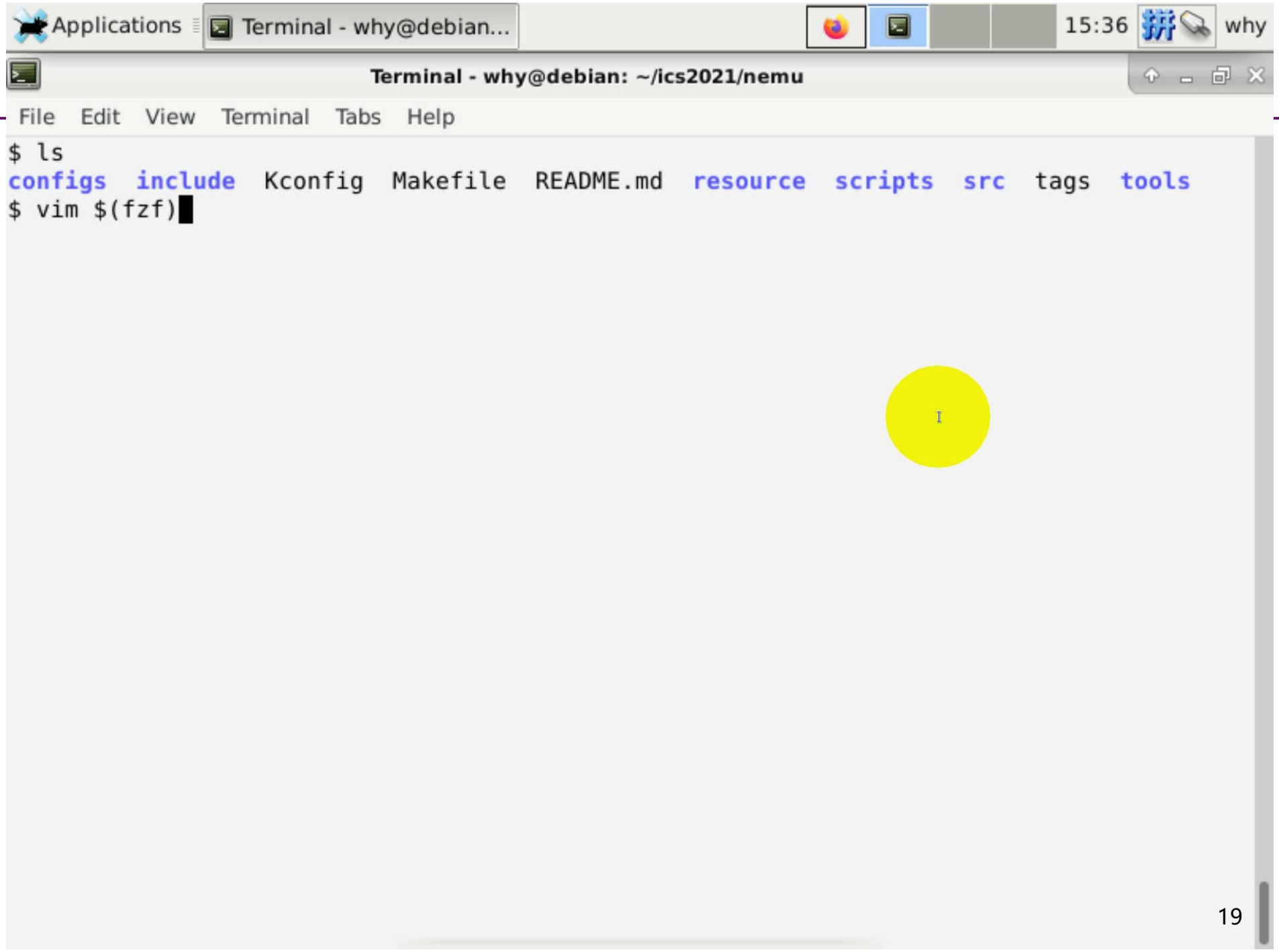
    - 浏览：:cn, :cp, ……

main()

- 比想象中短很多……

```c
int main(int argc, char *argv[]) {
  init_monitor(argc, argv);
  engine_start();
  return is_exit_status_bad();
}
```

```c
#include <common.h>

void init_monitor(int, char *[]);
void am_init_monitor();
void engine_start();


int is_exit_status_bad();

int main(int argc, char *argv[]) {
  /* Initialize the monitor. */
#ifdef CONFIG_TARGET_AM
  am_init_monitor();
#else
  init_monitor(argc, argv);
#endif

  /* Start engine. */
  engine_start();

  return is_exit_status_bad();
}
```


Applications
Terminal - why@debian...
15:36
why
Terminal - why@debian: ~/ics2021/nemu
File Edit View Terminal Tabs Help
$ ls
configs include Kconfig Makefile README.md resource scripts src tags tools
$ vim $(fzf)

main()

- 比想象中短很多……

```c
int main(int argc, char *argv[]) {
  init_monitor(argc, argv);
  engine_start();
  return is_exit_status_bad();
}
```

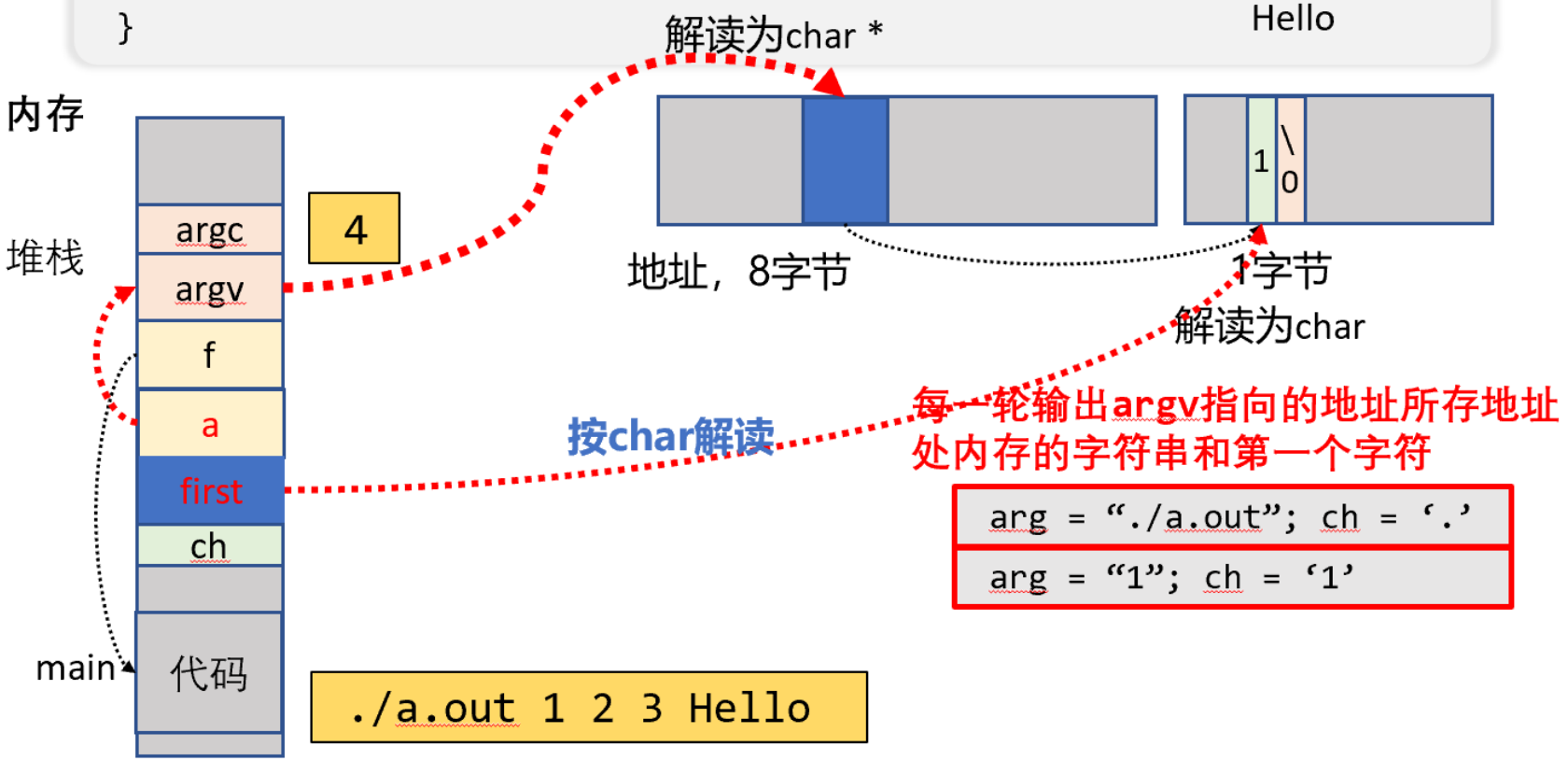
```c
#include <common.h>

void init_monitor(int, char *[]);
void am_init_monitor();
void engine_start();


int is_exit_status_bad();

int main(int argc, char *argv[]) {
  /* Initialize the monitor. */
#ifdef CONFIG_TARGET_AM
  am_init_monitor();
#else
  init_monitor(argc, argv);
#endif

  /* Start engine. */
  engine_start();

  return is_exit_status_bad();
}
```

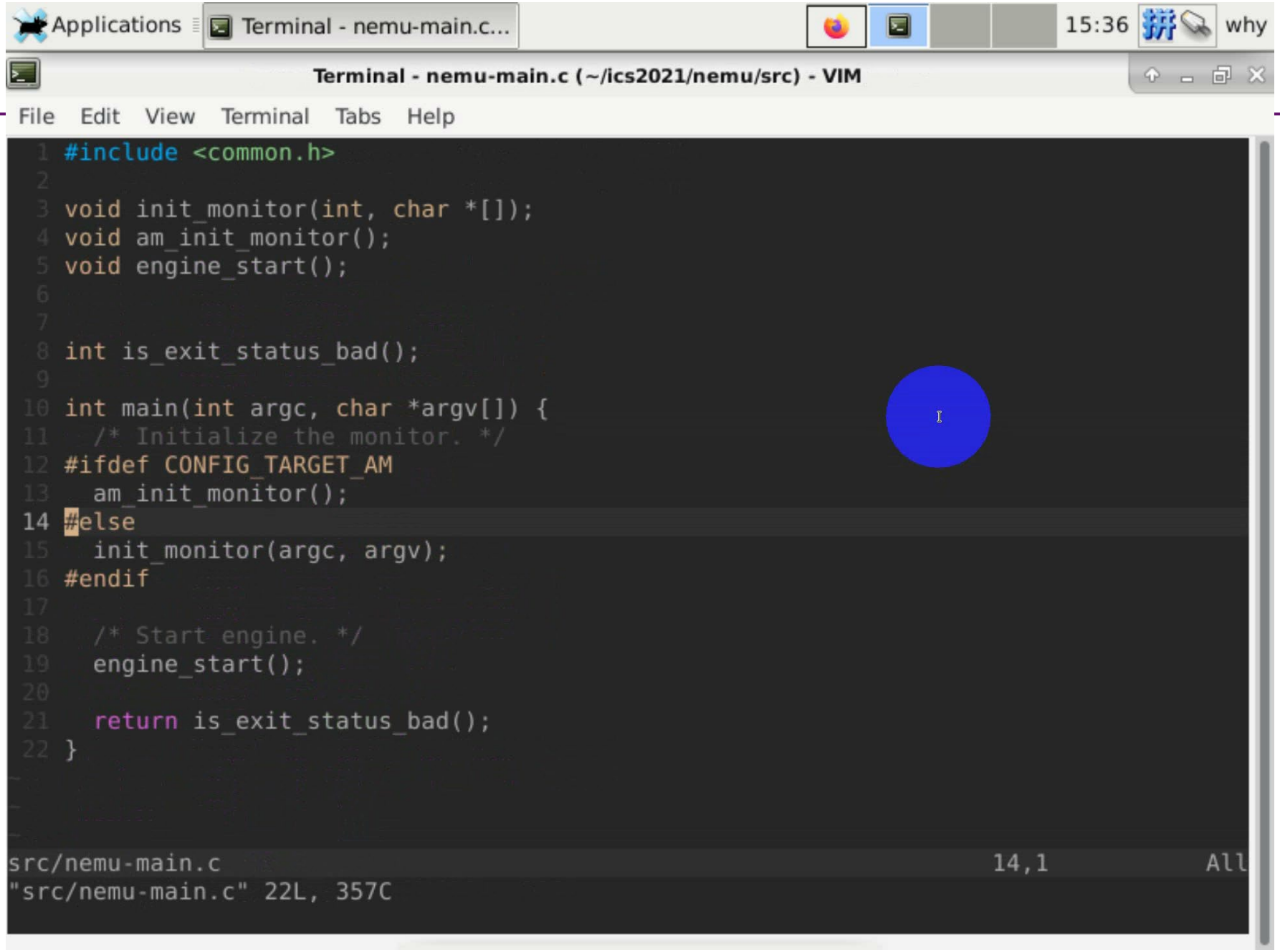
- Comments
  - 把 `argc, argv` 传递给另一个函数是 C 的 idiomatic use
  - `init_monitor` 代码在哪里?
    - 每次都 `grep` 效率太低
    - 需要更先进的工具 (稍候介绍)

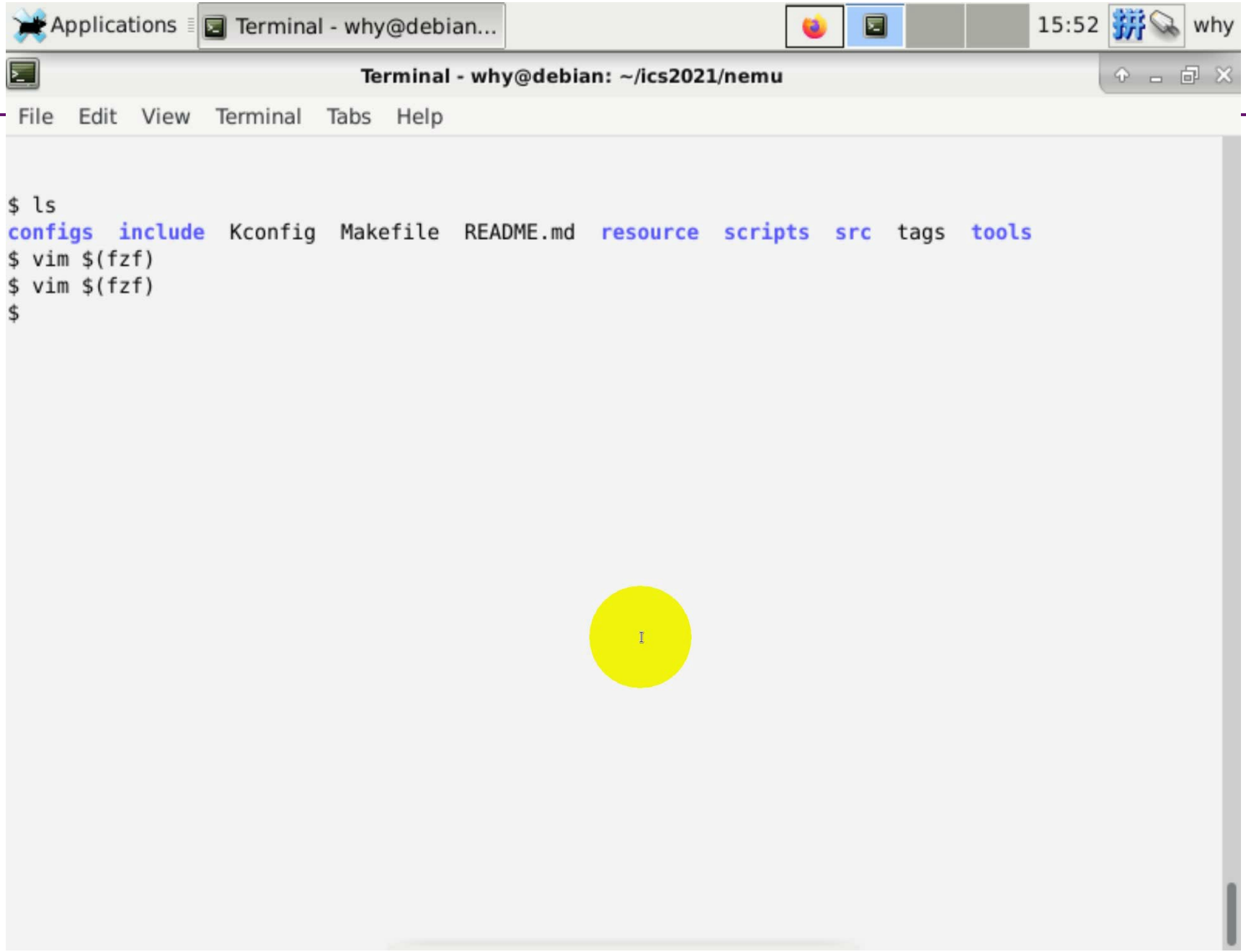
```
int main(int argc, char *argv[]) {
    int (*f)(int, char *[]) = main;
    if (argc != 0) {
        char ***a = &argv, *first = argv[0
        printf("arg = \"%s\"; ch = '%c'\n
        assert(***a == ch);
        f(argc - 1, argv + 1);
    }
}
```

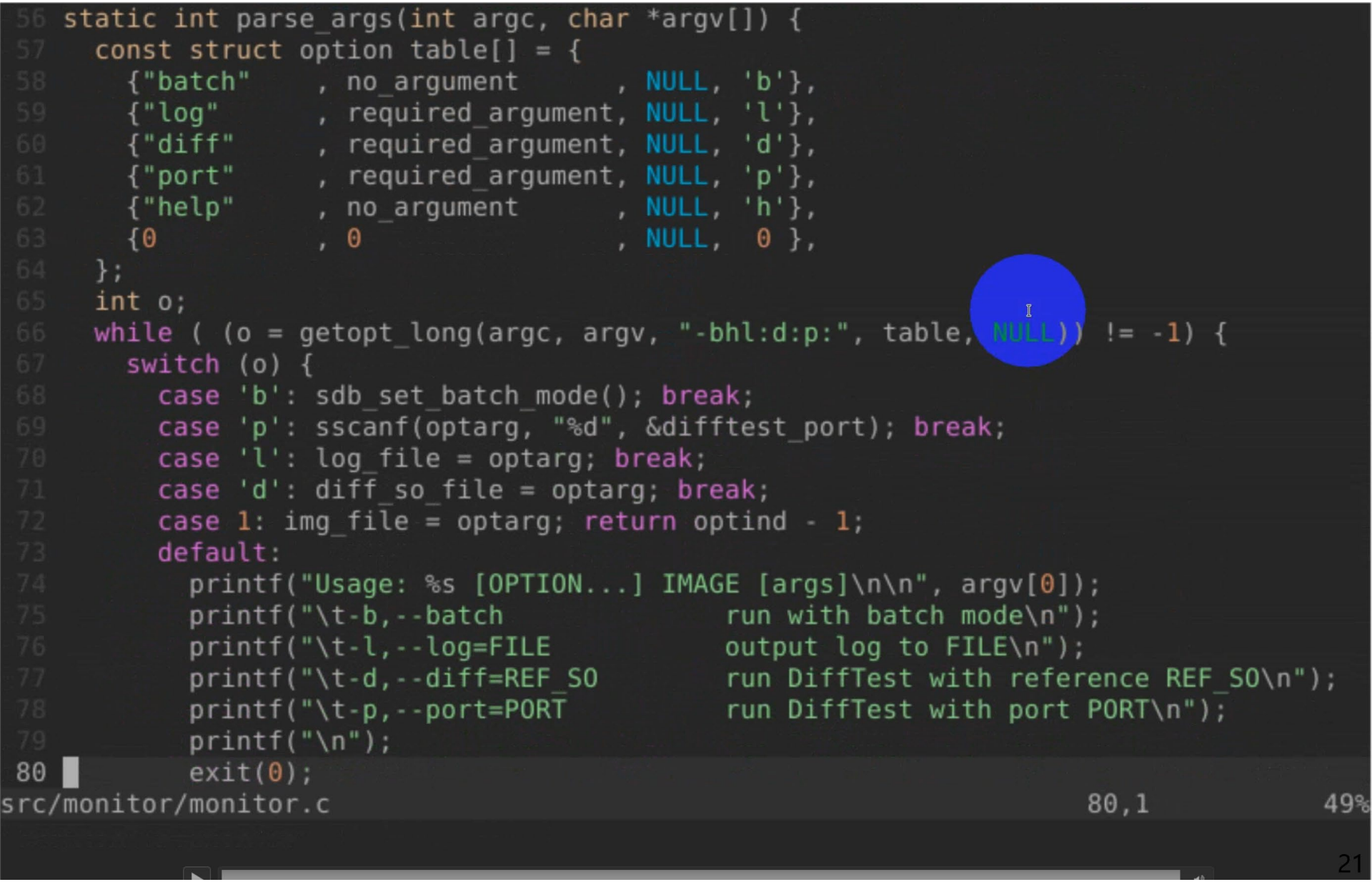
```
$ ./a.out 1 2 3 hello
arg = "./a.out";  ch = '.'
arg = "1";  ch = '1'
arg = "2";  ch = '2'
arg = "3";  ch = '3'
arg = "hello";  ch = 'h'
```

Hello




Applications
Terminal - nemu-main.c...
15:36
why
Terminal - nemu-main.c (~/ics2021/nemu/src) - VIM
File Edit View Terminal Tabs Help
1 #include <common.h>
2
3 void init_monitor(int, char *[]);
4 void am_init_monitor();
5 void engine_start();
6
7
8 int is_exit_status_bad();
9
10 int main(int argc, char *argv[]) {
11 /* Initialize the monitor. */
12 #ifdef CONFIG_TARGET_AM
13 am_init_monitor();
14 #else
15 init_monitor(argc, argv);
16 #endif
17
18 /* Start engine. */
19 engine_start();
20
21 return is_exit_status_bad();
22 }
src/nemu-main.c 14,1 All
"src/nemu-main.c" 22L, 357C


Applications
Terminal - why@debian...
15:52
why
Terminal - why@debian: ~/ics2021/nemu
File Edit View Terminal Tabs Help
$ ls
configs include Kconfig Makefile README.md resource scripts src tags tools
$ vim $(fzf)
$ vim $(fzf)
$


static int parse_args(int argc, char *argv[]) {
const struct option table[] = {
{"batch" , no_argument , NULL, 'b'},
{"log" , required_argument, NULL, 'l'},
{"diff" , required_argument, NULL, 'd'},
{"port" , required_argument, NULL, 'p'},
{"help" , no_argument , NULL, 'h'},
{0 , 0 , NULL, 0 },
};
int o;
while ((o = getopt_long(argc, argv, "-bhl:d:p:", table, NULL)) != -1) {
switch (o) {
case 'b': sdb_set_batch_mode(); break;
case 'p': sscanf(optarg, "%d", &difftest_port); break;
case 'l': log_file = optarg; break;
case 'd': diff_so_file = optarg; break;
case 1: img_file = optarg; return optind - 1;
default:
printf("Usage: %s [OPTION...] IMAGE [args]\n\n", argv[0]);
printf("\t-b,--batch run with batch mode\n");
printf("\t-l,--log=FILE output log to FILE\n");
printf("\t-d,--diff=REF_SO run DiffTest with reference REF_SO\n");
printf("\t-p,--port=PORT run DiffTest with port PORT\n");
printf("\n");
80
exit(0);
src/monitor/monitor.c
80,1
49%

- `make run`

```
$ make run
+ LD /home/why/ics2021/nemu/build/riscv32-nemu-interpreter
/home/why/ics2021/nemu/build/riscv32-nemu-interpreter --log=/home/why/ics2021/nemu/build/nemu-log.
txt
[src/memory/paddr.c:35 init_mem] physical memory area [0x80000000, 0x88000000]
[src/monitor/monitor.c:36 load_img] No image is given. Use the default build-in image.
[src/monitor/monitor.c:12 welcome] Debug: ON
[src/monitor/monitor.c:16 welcome] If debug mode is on, a log file will be generated to record eve
ry instruction NEMU executes. This may lead to a large log file. If it is not necessary, you can t
urn it off in include/common.h.
[src/monitor/monitor.c:17 welcome] Build time: 15:57:12, Aug 30 2021
Welcome to riscv32-NEMU!
For help, type "help"
[src/monitor/monitor.c:20 welcome] Exercise: Please remove me in the source code and compile NEMU
again.
(nemu) help
help - Display informations about all supported commands
c - Continue the execution of the program
q - Exit NEMU
(nemu)
```

- `./build/riscv32-nemu-interpreter`

```
$ ./build/riscv32-nemu-interpreter
[src/memory/paddr.c:35 init_mem] physical memory area [0x80000000, 0x88000000]
[src/monitor/monitor.c:36 load_img] No image is given. Use the default build-in image.
[src/monitor/monitor.c:12 welcome] Debug: ON
[src/monitor/monitor.c:16 welcome] If debug mode is on, a log file will be generated to record eve
ry instruction NEMU executes. This may lead to a large log file. If it is not necessary, you can t
urn it off in include/common.h.
[src/monitor/monitor.c:17 welcome] Build time: 15:57:12, Aug 30 2021
Welcome to riscv32-NEMU!
For help, type "help"
[src/monitor/monitor.c:20 welcome] Exercise: Please remove me in the source code and compile NEMU
again.
(nemu)
```

parse_args()

- 这个函数的名字起的很好，看了就知道要做什么
  - 满足好代码不言自明的特性
  - 的确是用来解析命令行参数的，`-b`, `-l`, `...`
  - 使用了 `getopt` → RTFM!

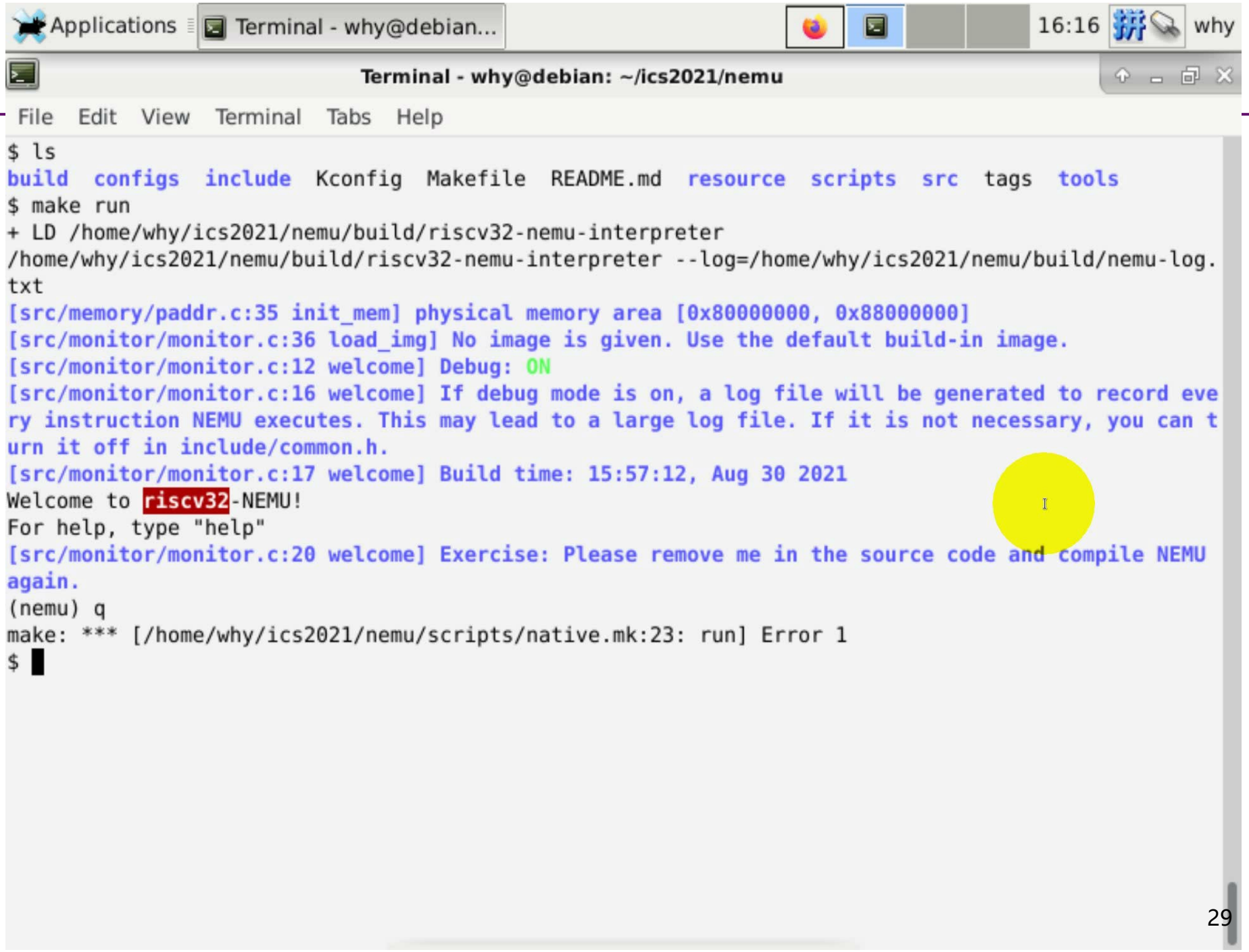
- 失败的尝试：`man getopt` → `getopt (1)`

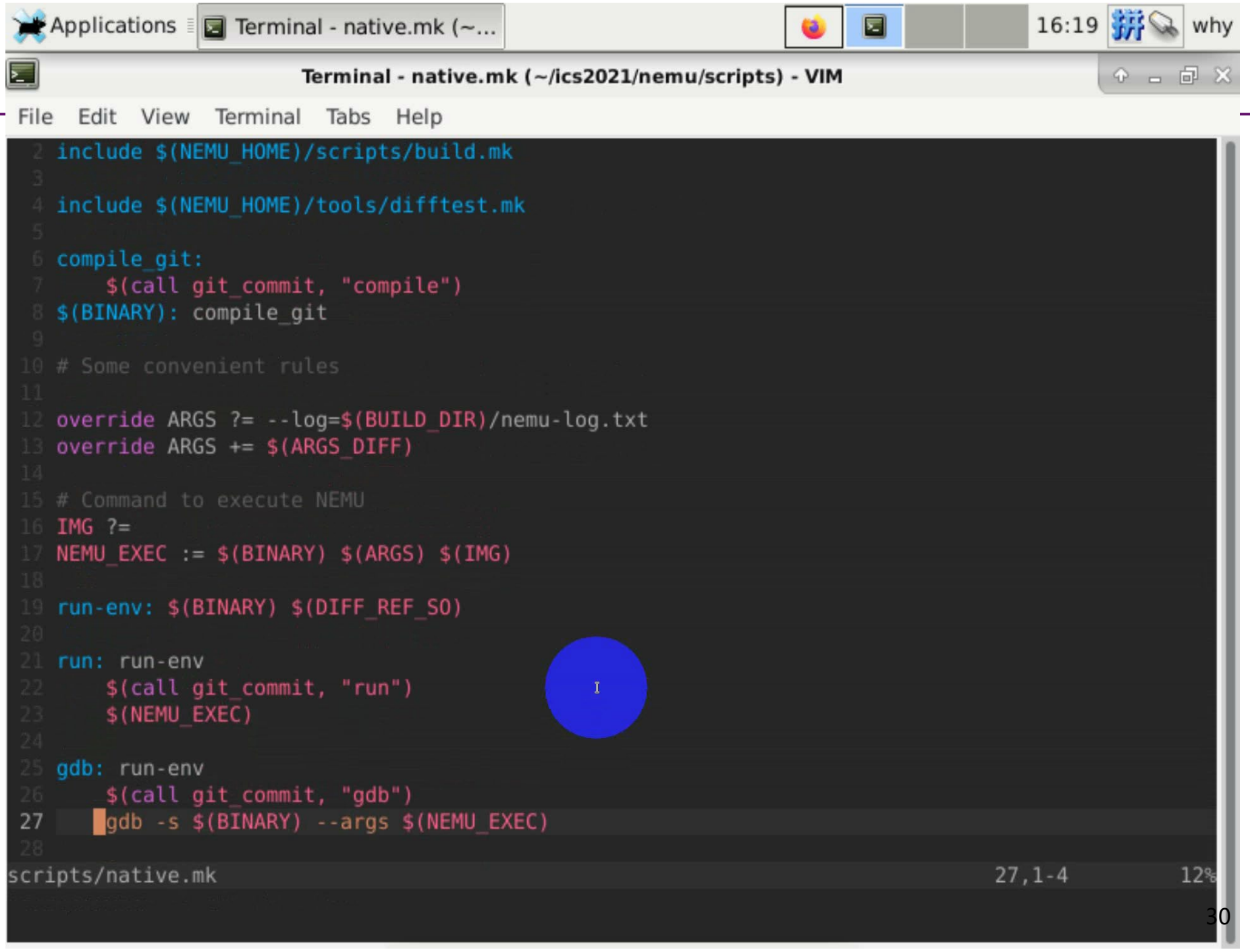
- 成功的尝试
  - 捷径版：STFW “`C getopt`” → 网页/博客/...
  - 专业版：`man -k getopt` → `man 3 getopt`

- 意外之喜：`man` 还送了个例子！跟 `parse_args` 的用法一样耶

```
$ ./build/riscv32-nemu-interpreter --help
Usage: ./build/riscv32-nemu-interpreter [OPTION...] IMAGE [args]

        -b,--batch              run with batch mode
        -l,--log=FILE           output log to FILE
        -d,--diff=REF_SO        run DiffTest with reference REF_SO
        -p,--port=PORT          run DiffTest with port PORT

$ ./build/riscv32-nemu-interpreter -b
[src/memory/paddr.c:34 init_mem] physical memory area [0x80000000, 0x88000000]
[src/monitor/monitor.c:36 load_img] No image is given. Use the default build-in image.
[src/monitor/monitor.c:12 welcome] Debug: ON
[src/monitor/monitor.c:13 welcome] If debug mode is on, a log file will be generated to record every ins
truction NEMU executes. This may lead to a large log file. If it is not necessary, you can turn it off i
n include/common.h.
[src/monitor/monitor.c:17 welcome] Build time: 10:17:25, Sep 23 2021
Welcome to riscv32-NEMU!
For help, type "help"
[src/cpu/cpu-exec.c:104 cpu_exec] nemu: HIT GOOD TRAP at pc = 0x8000000c
[src/cpu/cpu-exec.c:69 monitor_statistic] host time spent = 6 us
[src/cpu/cpu-exec.c:70 monitor_statistic] total guest instructions = 4
[src/cpu/cpu-exec.c:71 monitor_statistic] simulation frequency = 666,666 instr/s
$
```

怎么给nemu传指令?


Applications
Terminal - why@debian...
16:16
why
Terminal - why@debian: ~/ics2021/nemu
File Edit View Terminal Tabs Help
$ ls
build configs include Kconfig Makefile README.md resource scripts src tags tools
$ make run
+ LD /home/why/ics2021/nemu/build/riscv32-nemu-interpreter
/home/why/ics2021/nemu/build/riscv32-nemu-interpreter --log=/home/why/ics2021/nemu/build/nemu-log.txt
[src/memory/paddr.c:35 init_mem] physical memory area [0x80000000, 0x88000000]
[src/monitor/monitor.c:36 load_img] No image is given. Use the default build-in image.
[src/monitor/monitor.c:12 welcome] Debug: ON
[src/monitor/monitor.c:16 welcome] If debug mode is on, a log file will be generated to record every instruction NEMU executes. This may lead to a large log file. If it is not necessary, you can turn it off in include/common.h.
[src/monitor/monitor.c:17 welcome] Build time: 15:57:12, Aug 30 2021
Welcome to riscv32-NEMU!
For help, type "help"
[src/monitor/monitor.c:20 welcome] Exercise: Please remove me in the source code and compile NEMU again.
(nemu) q
make: *** [/home/why/ics2021/nemu/scripts/native.mk:23: run] Error 1
$


Applications
Terminal - native.mk (~...
16:19
why
Terminal - native.mk (~/ics2021/nemu/scripts) - VIM
File Edit View Terminal Tabs Help
include $(NEMU_HOME)/scripts/build.mk
include $(NEMU_HOME)/tools/difftest.mk
compile_git:
$(call git_commit, "compile")
$(BINARY): compile_git
Some convenient rules
override ARGS ?= --log=$(BUILD_DIR)/nemu-log.txt
override ARGS += $(ARGS_DIFF)
Command to execute NEMU
IMG ?=
NEMU_EXEC := $(BINARY) $(ARGS) $(IMG)
run-env: $(BINARY) $(DIFF_REF_SO)
run: run-env
$(call git_commit, "run")
$(NEMU_EXEC)
gdb: run-env
$(call git_commit, "gdb")
gdb -s $(BINARY) --args $(NEMU_EXEC)
scripts/native.mk
27,1-4
12%

NEMU：一个命令行工具

- The friendly source code
  - 命令行可以控制 NEMU 的行为
  - 我们甚至看到了 `--help` 帮助信息

- 如何让我们的 NEMU 打印它?
  - 问题等同于：`make run` 到底做了什么
  - 方法 1: 阅读 `Makefile`
  - 方法 2: 借助 `GNU Make` 的 `-n` 选项

开始痛苦的代码阅读之旅：坚持！

代码选讲

```c
static int parse_args(int argc, char *argv[]) {
  const struct option table[] = {
    {"batch"    , no_argument      , NULL, 'b'},
    {"log"      , required_argument, NULL, 'l'},
    {"diff"     , required_argument, NULL, 'd'},
    {"port"     , required_argument, NULL, 'p'},
    {"help"     , no_argument      , NULL, 'h'},
    {0          , 0                , NULL,  0 },
  };
  int o;
  while ( (o = getopt_long(argc, argv, "-bhl:d:p:", table, NULL)) != -1) {
    switch (o) {
      case 'b': sdb_set_batch_mode(); break;
      case 'p': sscanf(optarg, "%d", &difftest_port); break;
      case 'l': log_file = optarg; break;
      case 'd': diff_so_file = optarg; break;
      case 1: img_file = optarg; return optind - 1;
      default:
        printf("Usage: %s [OPTION...] IMAGE [args]\n\n", argv[0]);
        printf("\t-b,--batch              run with batch mode\n");
        printf("\t-l,--log=FILE           output log to FILE\n");
        printf("\t-d,--diff=REF_SO        run DiffTest with reference REF_SO\n");
        printf("\t-p,--port=PORT          run DiffTest with port PORT\n");
        printf("\n");
        exit(0);
    }
  }
  return 0;
}
```

src/monitor/monitor.c 85,0-1

static

```
static int parse_args(int argc, char *argv[]) { ... }
```

- parse_args 函数是 static的，这是什么意思?
  - static (C99 6.2.2 #3): If the declaration of a file scope identifier for an object or a function contains the storage- class specifier static, the identifier has internal linkage.
- 可以是static，建议是static
- 告诉编译器符号不要泄露到文件 (*translation unit*) 之外

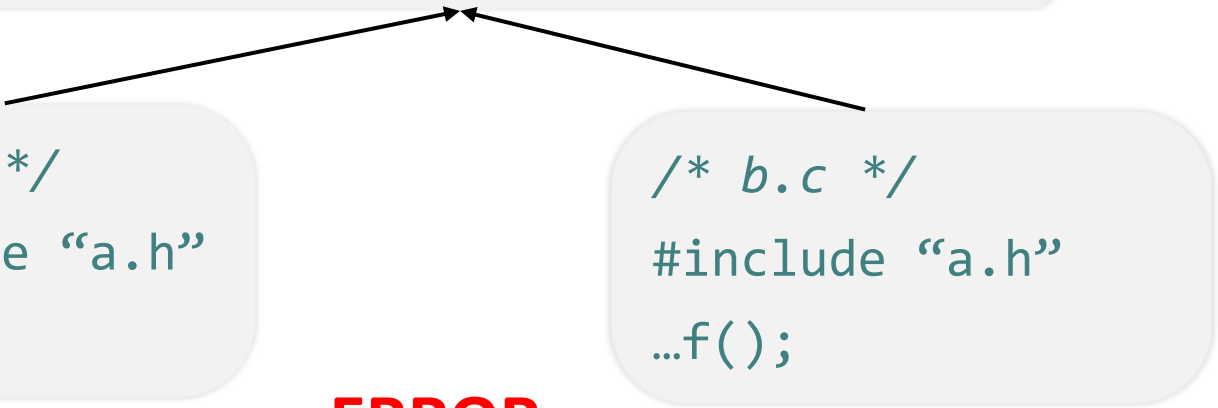
static

- 我们都知道，如果在两个文件里定义了重名的函数，能够分别编译，但链接会出错：

```
/* a.c */ int f() { return 0; }
/* b.c */ int f() { return 1; }
```

  - b.c:(.text+0x0): multiple definition of *f*; a.c:(.text+0xb): first defined here

```
/* a.h */ int f() { return 0; }
```



static

- 我们都知道，如果在两个文件里定义了重名的函数，能够分别编译，但链接会出错：

```
/* a.c */ int f() { return 0; }
/* b.c */ int f() { return 1; }
```

  - b.c:(.text+0x0): multiple definition of *f*; a.c:(.text+0xb): first defined here

```
/* a.h */ static int f() { return 0; }
```

```
/* a.c */
#include “a.h”
…f();
```

```
/* b.c */
#include “a.h”
…f();
```

Pass

static

- 我们都知道，如果在两个文件里定义了重名的函数，能够分别编译，但链接会出错：

```
/* a.c */ int f() { return 0; }
/* b.c */ int f() { return 1; }
```

  - b.c:(.text+0x0): multiple definition of *f*; a.c:(.text+0xb): first defined here

```
/* a.h */ static int f() { return 0; }
          // static int a;
```

```
/* a.c */
#include “a.h”
…f();
```

```
/* b.c */
#include “a.h”
…f();
```

```
/* x.c */
#include “a.h”
```

static

- 我们都知道，如果在两个文件里定义了重名的函数，能够分别编译，但链接会出错：

```
/* a.c */ int f() { return 0; }
/* b.c */ int f() { return 1; }
```

  - b.c:(.text+0x0): multiple definition of *f*; a.c:(.text+0xb): first defined here

```
/* a.h */ static int f() { return 0; }
          // static int a;
```

```
/* a.c */
#include “a.h”
…f();
```

```
/* b.c */
#include “a.h”
…f();
```

```
/* x.c */
#include “a.h”
```

WARNING

ERROR (-Wall -Werror)

更多思考

- 这也是为什么不在头文件里定义函数的原因
  - 两个 translation unit 同时引用，就导致 multiple definition

- 思考题：为什么 C++ 能把 class 都定义到头文件里？？？像 vector 的实现就是直接粘贴进去的

/riscv32//reg.h

```
 1 #ifndef __RISCV32_REG_H__
 2 #define __RISCV32_REG_H__
 3
 4 #include <common.h>
 5
 6 static inline int check_reg_idx(int idx) {
 7   IFDEF(CONFIG_RT_CHECK, assert(idx >= 0 && idx < 32));
 8   return idx;
 9 }
10
11 #define gpr(idx) (cpu.gpr[check_reg_idx(idx)]._32)
12
13 static inline const char* reg_name(int idx, int width) {
14   extern const char* regs[];
15   return regs[check_reg_idx(idx)];
16 }
17
18 #endif
```

static inline

- 如果你的程序较短且性能攸关，则可以使用 static inline 函数定义在头文件中。例子 (**/riscv32/**/reg.h)：

```
static inline int check_reg_index(int idx) {
        IFDEF(CONFIG_RT_CHECK, assert(idx >= 0 && index < 32));
        return idx;
}
```

```
/* a.h */ static inline int f() { return 0; }
          // static int a;
```

```
/* a.c */
#include “a.h”
…f();
```

```
/* b.c */
#include “a.h”
…f();
```

```
/* x.c */
#include “a.h”
```

static inline

- 如果你的程序较短且性能攸关，则可以使用 static inline 函数定义在头文件中。例子 (**/riscv32/**/reg.h)：

```
static inline int check_reg_index(int idx) {
        IFDEF(CONFIG_RT_CHECK, assert(idx >= 0 && index < 32));
        return idx;
}
```

- check_reg_index完全可以单独放在一个 C 文件里，头文件中只保留声明：

```
int check_reg_index(int index);
```

  - 但这样会导致在编译时，编译出一条额外的 call 指令 (假设没有 LTO)
  - 使用 inline 可以在调用 check_reg_index(0) 编译优化成零开销

assert

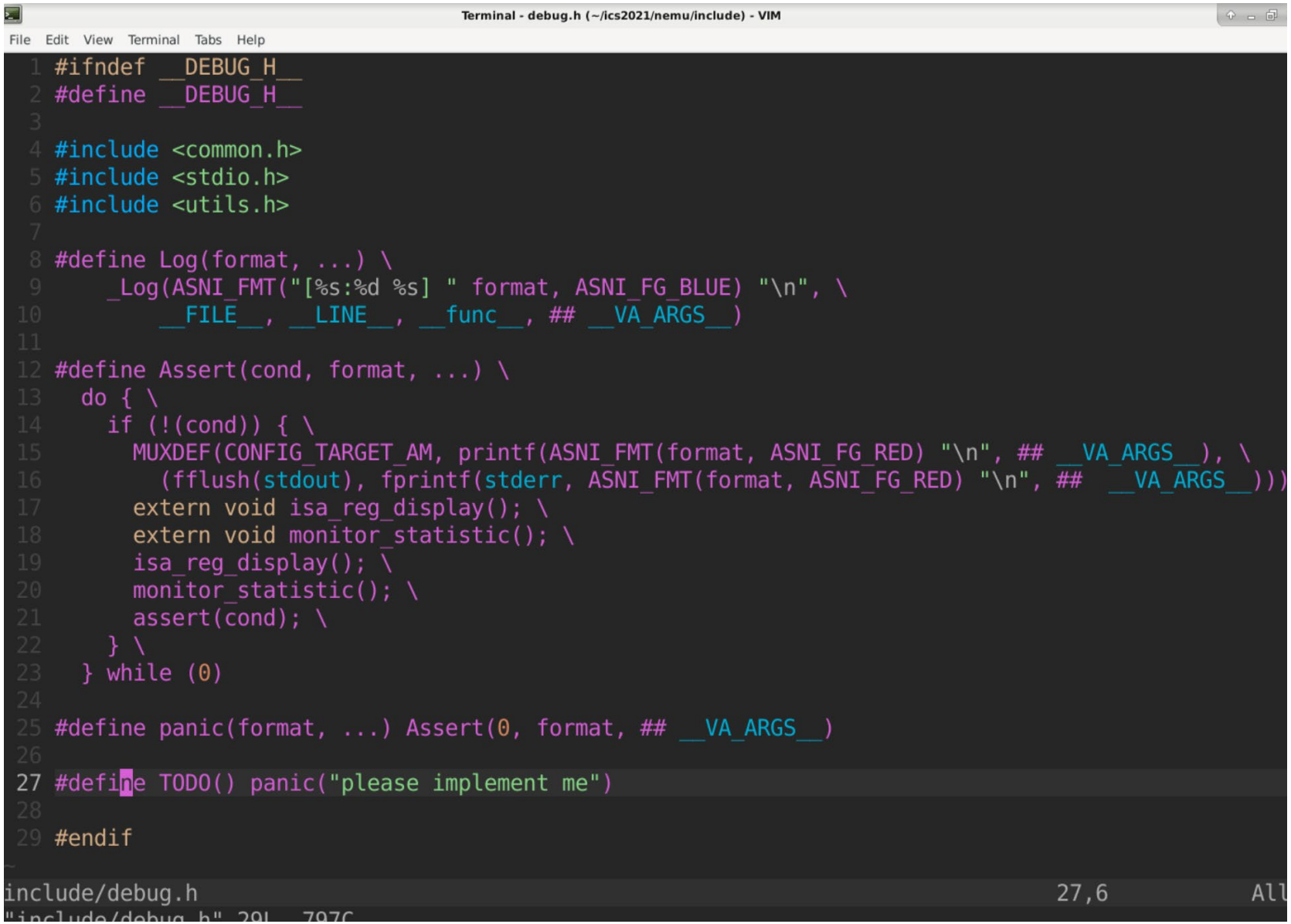
```
#define assert(cond) if (!(cond)) panic(...);
```

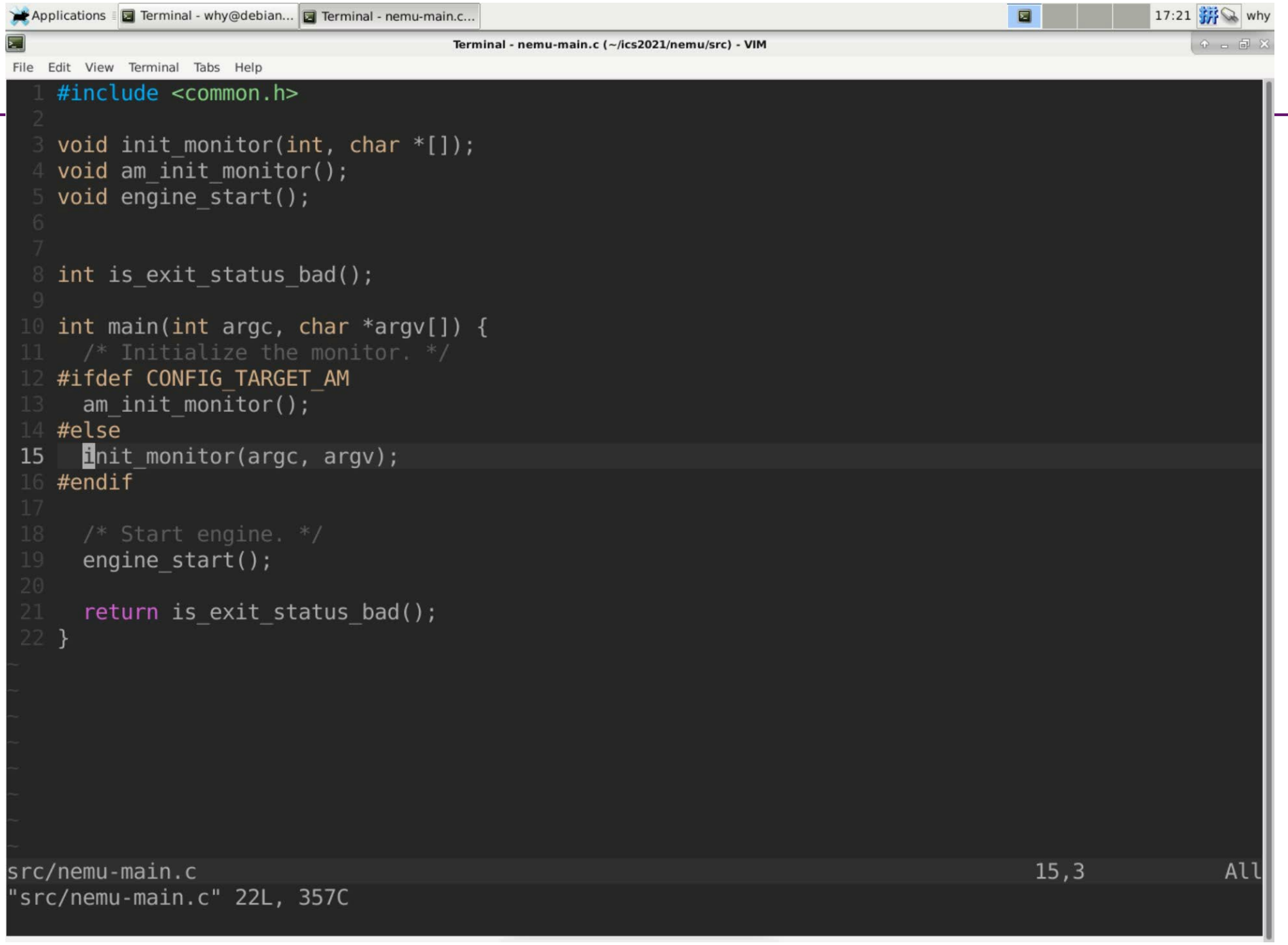
- 注意特殊情况

```
if (...) assert(0); // 上面的assert对么?
else ...
```

```
#define assert(cond) \ // nemu/**/debug.h
do { \
   if (!(cond)) { \
      fprintf(stderr, "Fail @ %s:%d", __FILE__, __LINE__);
\
      exit(1); \
   } \
} while (0)

#define assert(cond) ({ ... }) // GCC
```

**/debug.h


Terminal - debug.h (~/ics2021/nemu/include) - VIM
File Edit View Terminal Tabs Help
#ifndef __DEBUG_H__
#define __DEBUG_H__
#include <common.h>
#include <stdio.h>
#include <utils.h>
#define Log(format, ...) \
_Log(ASNI_FMT("[%s:%d %s] " format, ASNI_FG_BLUE) "\n", \
__FILE__, __LINE__, __func__, ## __VA_ARGS__)
#define Assert(cond, format, ...) \
do { \
if (!(cond)) { \
MUXDEF(CONFIG_TARGET_AM, printf(ASNI_FMT(format, ASNI_FG_RED) "\n", ## __VA_ARGS__), \
(fflush(stdout), fprintf(stderr, ASNI_FMT(format, ASNI_FG_RED) "\n", ## __VA_ARGS__)))
extern void isa_reg_display(); \
extern void monitor_statistic(); \
isa_reg_display(); \
monitor_statistic(); \
assert(cond); \
} \
} while (0)
#define panic(format, ...) Assert(0, format, ## __VA_ARGS__)
#define TODO() panic("please implement me")
#endif
include/debug.h 27,6 All

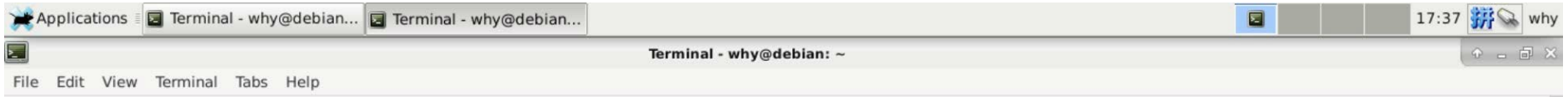

Applications
Terminal - why@debian...
Terminal - nemu-main.c...
17:21
why
Terminal - nemu-main.c (~/ics2021/nemu/src) - VIM
File Edit View Terminal Tabs Help
1 #include <common.h>
2
3 void init_monitor(int, char *[]);
4 void am_init_monitor();
5 void engine_start();
6
7
8 int is_exit_status_bad();
9
10 int main(int argc, char *argv[]) {
11 /* Initialize the monitor. */
12 #ifdef CONFIG_TARGET_AM
13 am_init_monitor();
14 #else
15 init_monitor(argc, argv);
16 #endif
17
18 /* Start engine. */
19 engine_start();
20
21 return is_exit_status_bad();
22 }
~
~
~
~
~
~
~
~
src/nemu-main.c 15,3 All
"src/nemu-main.c" 22L, 357C

千辛万苦……

- 之后的历程似乎就比较轻松了。有些东西不太明白(比如 `init_device()`)，但好像也不是很要紧，到了 `welcome()`

```
$ make run
+ LD /home/why/ics2021/nemu/build/riscv32-nemu-interpreter
/home/why/ics2021/nemu/build/riscv32-nemu-interpreter --log=/home/why/ics2021/nemu/build/nemu-log.t
xt
[src/memory/paddr.c:35 init_mem] physical memory area [0x80000000, 0x88000000]
[src/monitor/monitor.c:36 load_img] No image is given. Use the default build-in image.
[src/monitor/monitor.c:12 welcome] Debug: ON
[src/monitor/monitor.c:16 welcome] If debug mode is on, a log file will be generated to record ever
y instruction NEMU executes. This may lead to a large log file. If it is not necessary, you can tur
n it off in include/common.h.
[src/monitor/monitor.c:17 welcome] Build time: 17:23:31, Aug 30 2021
Welcome to riscv32-NEMU!
For help, type "help"
[src/monitor/monitor.c:20 welcome] Exercise: Please remove me in the source code and compile NEMU a
gain.
```

```
$ echo -e "\033[34m HAHA \033[0m"
 HAHA
$ echo -e "\033[43;34m HAHA \033[0m"
 HAHA
$ echo -e "\033[44;37m HAHA \033[0m"
 HAHA
```




```
$ ls
build  configs  include  in.txt  Kconfig  Makefile  README.md  resource  scripts  src  tags  tools
$
```

千辛万苦……

- 初始化终于完成
- 根本没碰到核心代码

理解代码：更进一步

来自以往同学的反馈

“我决定挑战自己，坚持在命令行中工作、使用 Vim 编辑代码。但在巨多的文件之间切换真是一言难尽。”

- 上手以后还在用 grep 找代码？
  - 你刚拿到项目的时候，grep 的确不错
  - 但如果你要一学期在这个项目上，效率就太低了
    - 曾经有无数的同学选择容忍这种低效率

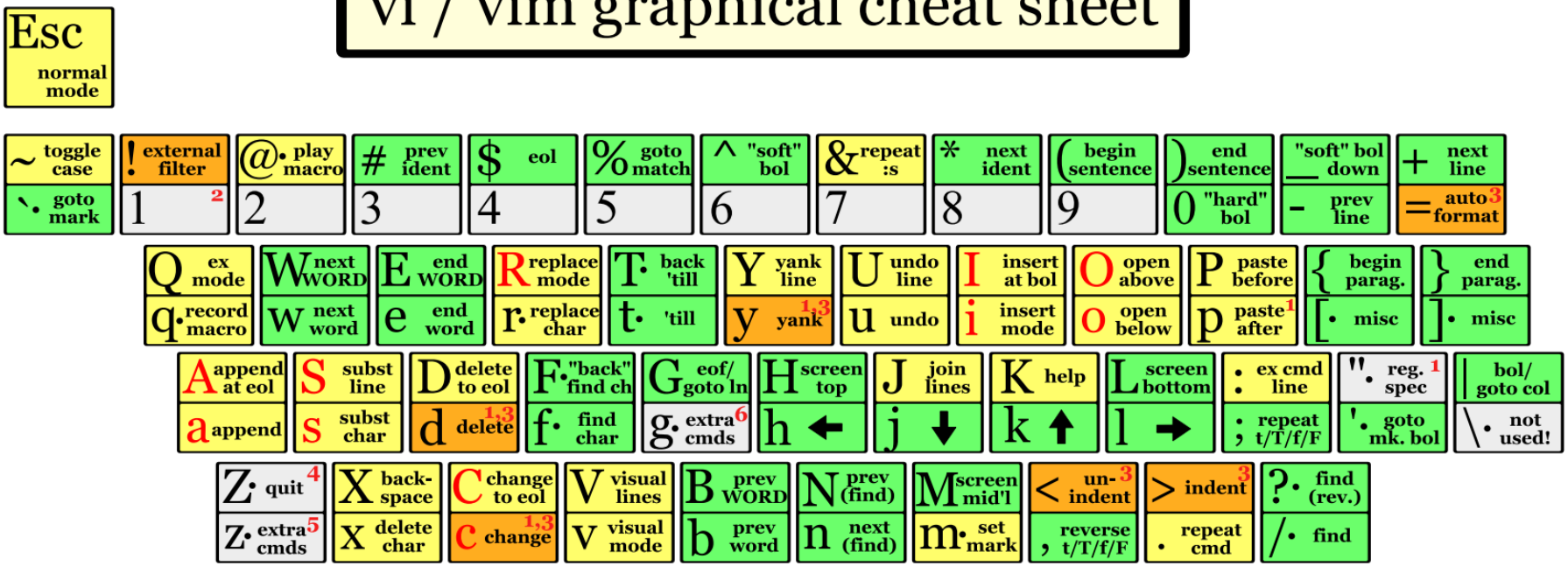
Vim：这都搞不定还引发什么编辑器圣战

- Marks (文件内标记)
  - ma, 'a, mA, 'A, ...
- Tags (在位置之间跳转)
  - :jumps, C-], C-i, C-o, :tjump, ...
- Tabs/Windows (管理多文件)
  - :tabnew, gt, gT, ...
- Folding (浏览大代码)
  - zc, zo, zR, ...

- 更多的功能/插件：(RTFM, STFW)
  - vimtutor

version 1.1
April 1st, 06

vi / vim graphical cheat sheet



motion — moves the cursor, or defines the range for an operator

command — direct action command, if red, it enters insert mode

operator — requires a motion afterwards, operates between cursor & destination

extra — special functions, requires extra input

q. — commands with a dot need a char argument afterwards

bol = beginning of line, eol = end of line,
mk = mark, yank = copy

words: `quux((foo, bar, baz);`
WORDs: `quux(foo, bar, baz);`

Main command line commands ('ex'):

:w (save), :q (quit), :q! (quit w/o saving)
:e f (open file f),
:%s/x/y/g (replace 'x' by 'y' filewide),
:h (help in vim), :new (new file in vim),

Other important comands:

CTRL-R: redo (vim),
CTRL-F/-B: page up/down,
CTRL-E/-Y: scroll line up/down,
CTRL-V: block-visual mode (vim only)

Visual mode:

Move around and type operator to act on selected region (vim only)

Notes:

(1) use "x before a yank/paste/del command to use that register ('clipboard') (x=a..z,*) (e.g.: "ay$ to copy rest of line to reg 'a')

(2) type in a number before any action to repeat it that number of times (e.g.: 2p, d2w, 5i, d4j)

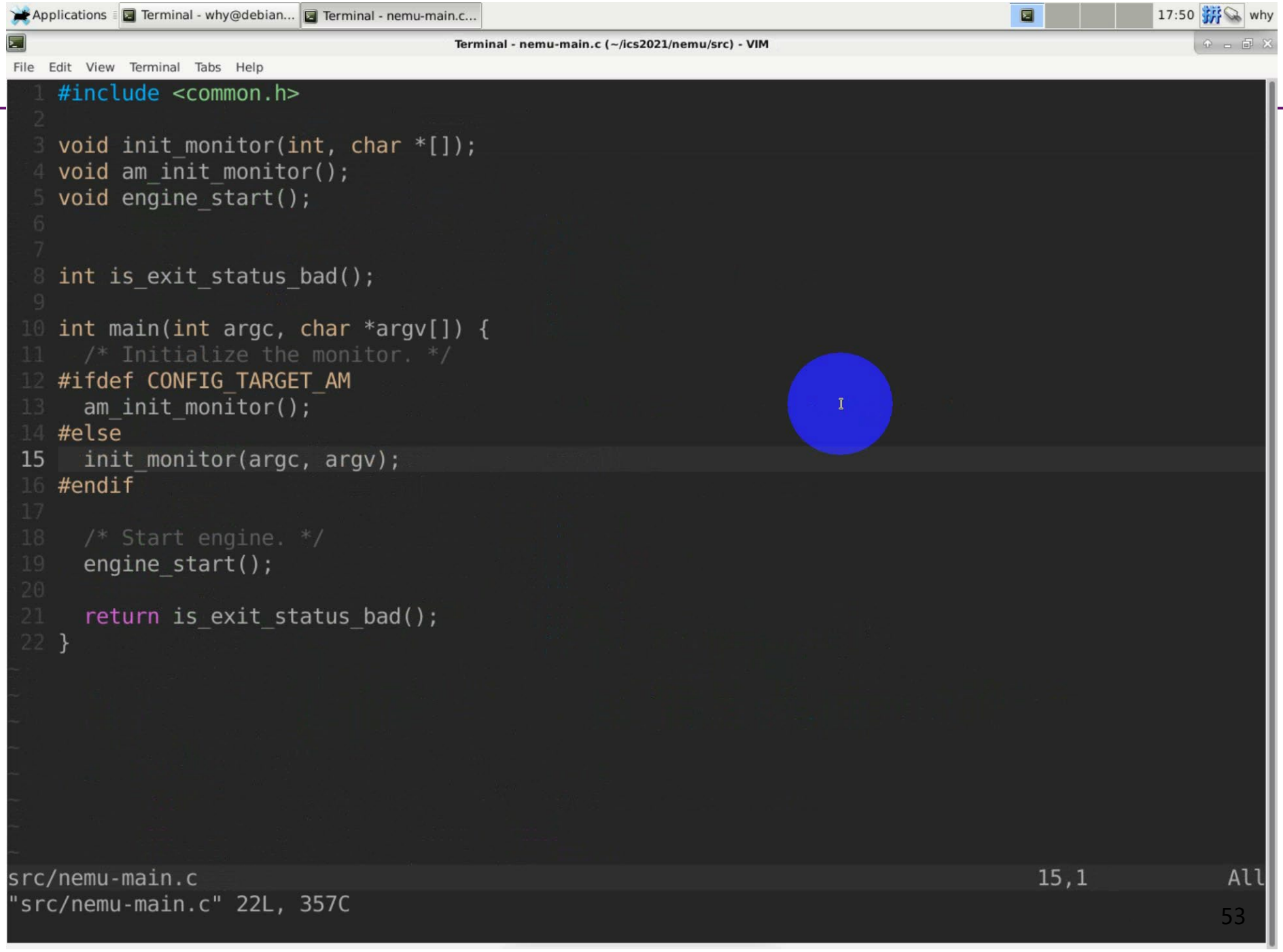
(3) duplicate operator to act on current line (dd = delete line, >> = indent line)

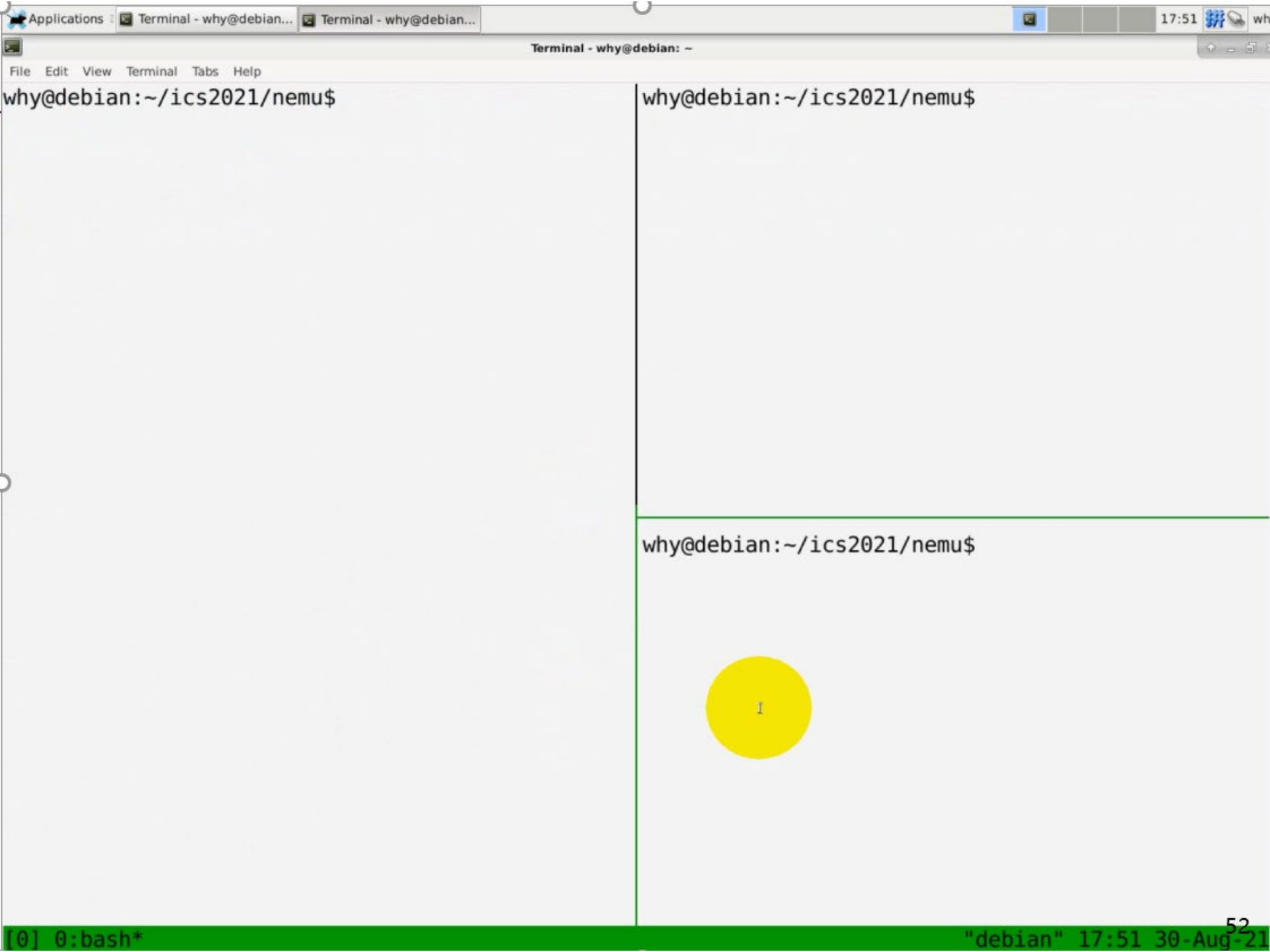
(4) ZZ to save & quit, ZQ to quit w/o saving

(5) zt: scroll cursor to top, zb: bottom, zz: center

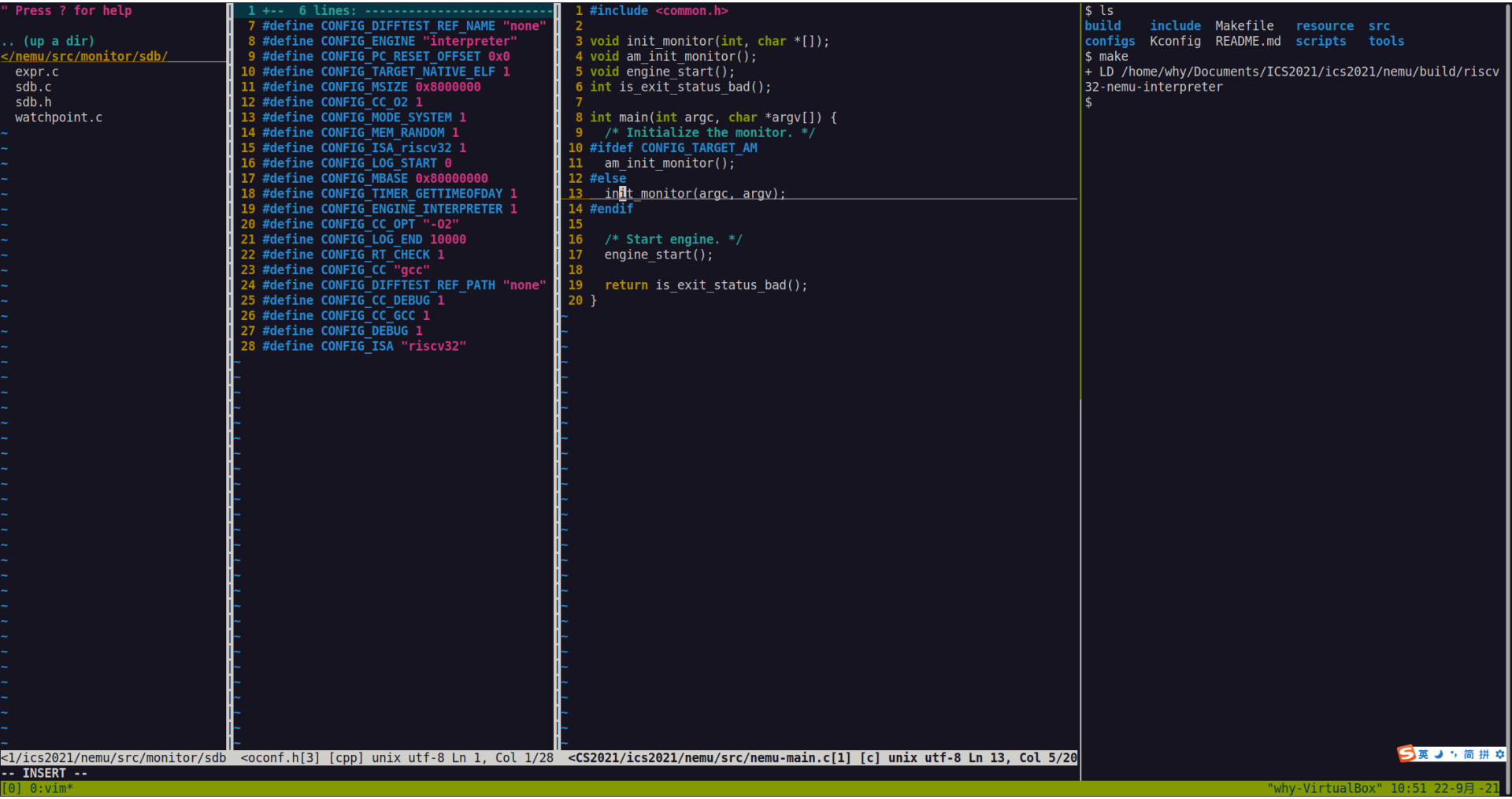
(6) gg: top of file (vim only), gf: open file under cursor (vim only)

For a graphical vi/vim tutorial & more tips, go to **www.viemu.com** - home of ViEmu, vi/vim emulation for Microsoft Visual Studio


Applications
Terminal - why@debian...
Terminal - nemu-main.c...
17:50
why
Terminal - nemu-main.c (~/ics2021/nemu/src) - VIM
File Edit View Terminal Tabs Help
1 #include <common.h>
2
3 void init_monitor(int, char *[]);
4 void am_init_monitor();
5 void engine_start();
6
7
8 int is_exit_status_bad();
9
10 int main(int argc, char *argv[]) {
11 /* Initialize the monitor. */
12 #ifdef CONFIG_TARGET_AM
13 am_init_monitor();
14 #else
15 init_monitor(argc, argv);
16 #endif
17
18 /* Start engine. */
19 engine_start();
20
21 return is_exit_status_bad();
22 }
~
~
~
~
~
~
~
~
~
src/nemu-main.c 15,1 All
"src/nemu-main.c" 22L, 357C


Applications
Terminal - why@debian...
Terminal - why@debian...
17:51
Terminal - why@debian: ~
File Edit View Terminal Tabs Help
why@debian:~/ics2021/nemu$
why@debian:~/ics2021/nemu$
why@debian:~/ics2021/nemu$
[0] 0:bash*
"debian" 17:51 30-Aug-21

如果你有块更大的屏幕


" Press ? for help
.. (up a dir)
</nemu/src/monitor/sdb/
expr.c
sdb.c
sdb.h
watchpoint.c
1 +-- 6 lines: ----------
7 #define CONFIG_DIFFTEST_REF_NAME "none"
8 #define CONFIG_ENGINE "interpreter"
9 #define CONFIG_PC_RESET_OFFSET 0x0
10 #define CONFIG_TARGET_NATIVE_ELF 1
11 #define CONFIG_MSIZE 0x8000000
12 #define CONFIG_CC_O2 1
13 #define CONFIG_MODE_SYSTEM 1
14 #define CONFIG_MEM_RANDOM 1
15 #define CONFIG_ISA_riscv32 1
16 #define CONFIG_LOG_START 0
17 #define CONFIG_MBASE 0x80000000
18 #define CONFIG_TIMER_GETTIMEOFDAY 1
19 #define CONFIG_ENGINE_INTERPRETER 1
20 #define CONFIG_CC_OPT "-O2"
21 #define CONFIG_LOG_END 10000
22 #define CONFIG_RT_CHECK 1
23 #define CONFIG_CC "gcc"
24 #define CONFIG_DIFFTEST_REF_PATH "none"
25 #define CONFIG_CC_DEBUG 1
26 #define CONFIG_CC_GCC 1
27 #define CONFIG_DEBUG 1
28 #define CONFIG_ISA "riscv32"
1 #include <common.h>
2
3 void init_monitor(int, char *[]);
4 void am_init_monitor();
5 void engine_start();
6 int is_exit_status_bad();
7
8 int main(int argc, char *argv[]) {
9 /* Initialize the monitor. */
10 #ifdef CONFIG_TARGET_AM
11 am_init_monitor();
12 #else
13 init_monitor(argc, argv);
14 #endif
15
16 /* Start engine. */
17 engine_start();
18
19 return is_exit_status_bad();
20 }
$ ls
build include Makefile resource src
configs Kconfig README.md scripts tools
$ make
+ LD /home/why/Documents/ICS2021/ics2021/nemu/build/riscv32-nemu-interpreter
$
<1/ics2021/nemu/src/monitor/sdb <oconf.h[3] [cpp] unix utf-8 Ln 1, Col 1/28 <CS2021/ics2021/nemu/src/nemu-main.c[1] [c] unix utf-8 Ln 13, Col 5/20
-- INSERT --
[0] 0:vim*
"why-VirtualBox" 10:51 22-9月 -21

VSCode: 现代工具来一套?

- 刚拿到手，VSCode 的体验并不是非常好
  - 满屏的红线/蓝线
  - 因为 Code 并知道 NEMU 是怎么编译的
  - IDE “编译运行” 背后没有魔法

- 另一方面，这些东西一定是可以配置的
  - 配置解析选项: `c_cpp_properties.json`
    - 解锁正确的代码解析
  - 配置构建选项: `tasks.json`
    - 解锁 `make` (可跟命令行参数)
  - 配置运行选项: `launch.json`
    - 解锁单步调试 (我们并不非常推荐单步调试)

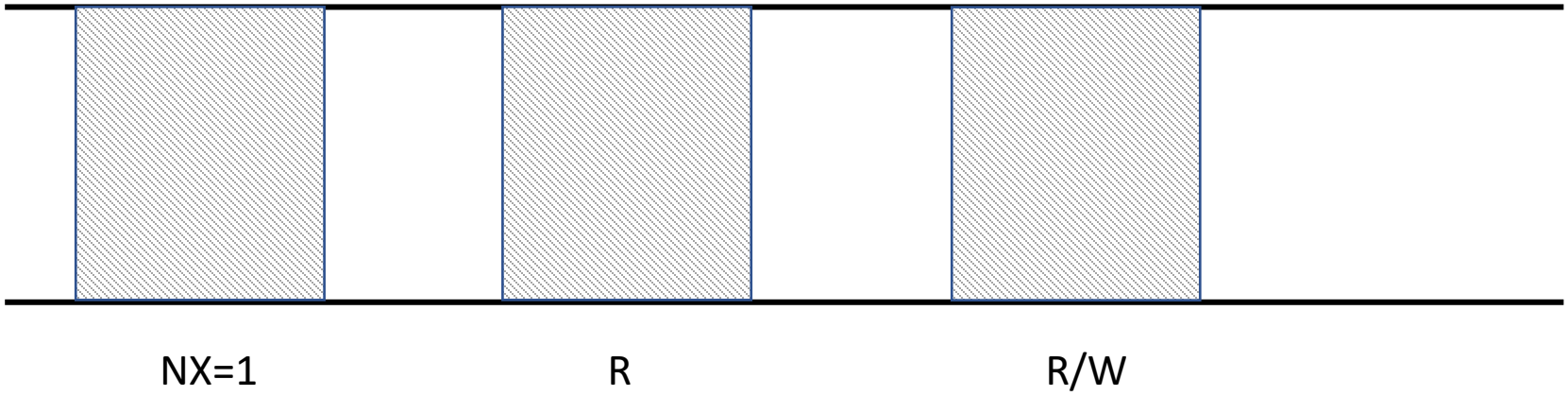
插入福利：调试Segmentation Fault

- 听说你的程序又 Segmentation Fault 了？
  - 百度 Segmentation Fault 得到的回答的解释可能是完全错误的

- 正确的解释
  - 指令越权访问内存 (r/w/x)
    - 原因很多，数组越界、memory corruption, ...
  - 指令未被执行，进程收到 SIGSEGV 信号
    - 默认的信号处理程序会 core dump 退出
    - Gdb调试：`backtrace`

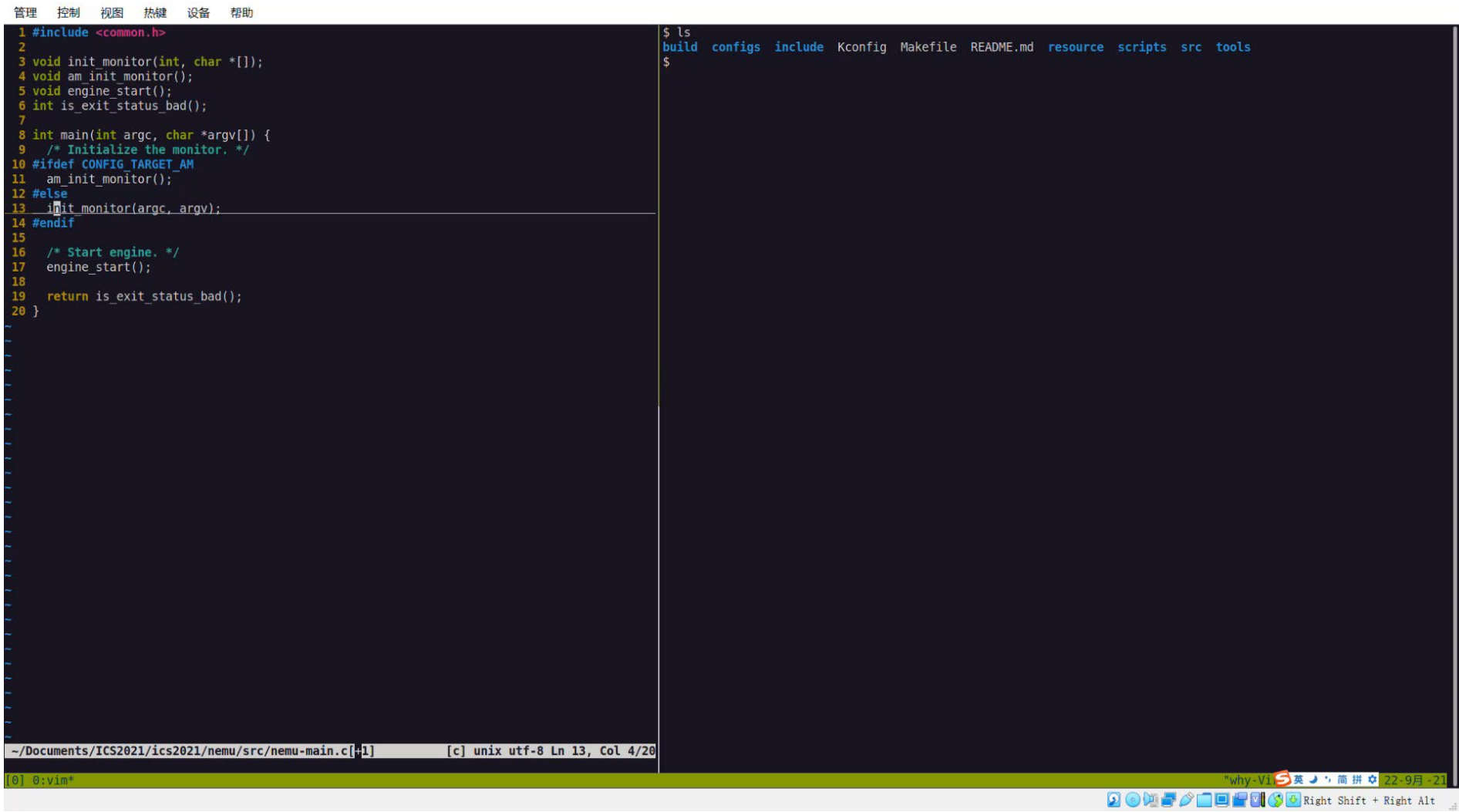
Segmentation Fault

- `*((int *) 0) = 0;`



- 指令越权访问内存 (r/w/x)

- SIGSEGV
  - `Core dumped`
  - `Gdb调试: backtrace`


管理 控制 视图 热键 设备 帮助
1 #include <common.h>
2
3 void init_monitor(int, char *[]);
4 void am_init_monitor();
5 void engine_start();
6 int is_exit_status_bad();
7
8 int main(int argc, char *argv[]) {
9 /* Initialize the monitor. */
10 #ifdef CONFIG_TARGET_AM
11 am_init_monitor();
12 #else
13 init_monitor(argc, argv);
14 #endif
15
16 /* Start engine. */
17 engine_start();
18
19 return is_exit_status_bad();
20 }
~/Documents/ICS2021/ics2021/nemu/src/nemu-main.c[+1] [c] unix utf-8 Ln 13, Col 4/20
$ ls
build configs include Kconfig Makefile README.md resource scripts src tools
$
[0] 0:vim*
"why-Vi
22-9月-21
Right Shift + Right Alt

好的编辑器：有时候也许不是万能的

- 又爱又恨的元编程
- 办法 1: RTFM + RTFSC + 写小程序尝试
- 办法 2: 预编译以后的代码应该好理解！
  - 还记得我们对 Makefile 的导读吗？
    - (说的容易做得难。直接 gcc -E 不是编译错误吗……)

```
#define def_INSTR_IDTABW(pattern, id, tab, width) \
  def_INSTR_raw(pattern_decode, pattern, \
      { concat(decode_, id)(s, width); return concat(table_, tab)(s); })
#define def_INSTR_IDTAB(pattern, id, tab)    def_INSTR_IDTABW(pattern, id, tab, 0)
#define def_INSTR_TABW(pattern, tab, width) def_INSTR_IDTABW(pattern, empty, tab, width)
#define def_INSTR_TAB(pattern, tab)         def_INSTR_IDTABW(pattern, empty, tab, 0)

#define def_hex_INSTR_IDTABW(pattern, id, tab, width) \
  def_INSTR_raw(pattern_decode_hex, pattern, \
      { concat(decode_, id)(s, width); return concat(table_, tab)(s); })
#define def_hex_INSTR_IDTAB(pattern, id, tab)    def_hex_INSTR_IDTABW(pattern, id, tab, 0)
#define def_hex_INSTR_TABW(pattern, tab, width) def_hex_INSTR_IDTABW(pattern, empty, tab, width)
#define def_hex_INSTR_TAB(pattern, tab)         def_hex_INSTR_IDTABW(pattern, empty, tab, 0)
```

```
// --- pattern matching wrappers for decode ---
#define INSTPAT(pattern, ...) do { \
  uint64_t key, mask, shift; \
  pattern_decode(pattern, STRLEN(pattern), &key, &mask, &shift); \
  if (((INSTPAT_INST(s) >> shift) & mask) == key) { \
    INSTPAT_MATCH(s, ##__VA_ARGS__); \
    goto *(__instpat_end); \
  } \
} while (0)

#define INSTPAT_START(name) { const void **__instpat_end = &&concat(__instpat_end_, name);
#define INSTPAT_END(name)   concat(__
```

```
#define src1R(n) do { *src1 = R(n); } while (0)
#define src2R(n) do { *src2 = R(n); } while (0)
#define destR(n) do { *dest = n; } while (0)
#define src1I(i) do { *src1 = i; } while (0)
#define src2I(i) do { *src2 = i; } while (0)
#define destI(i) do { *dest = i; } while (0)
```

```
#define INSTPAT_INST(s) ((s)->isa.inst
#define INSTPAT_MATCH(s, name, type, .
  decode_operand(s, &dest, &src1, &src
  __VA_ARGS__ ; \
}

  INSTPAT_START();
  INSTPAT("??????? ????? ????? ??? ????? 01101 11", lui    , U, R(dest) = src1);
  INSTPAT("??????? ????? ????? 010 ????? 00000 11", lw     , I, R(dest) = Mr(src1 + src2, 4)
);
  INSTPAT("??????? ????? ????? 010 ????? 01000 11", sw     , S, Mw(src1 + dest, 4, src2));

  INSTPAT("0000000 00001 00000 000 00000 11100 11", ebreak , N, NEMUTRAP(s->pc, R(10))); //
R(10) is $a0
  INSTPAT("??????? ????? ????? ??? ????? ????? ??", inv    , N, INV(s->pc));
  INSTPAT_END();

  R(0) = 0; // reset $zero to 0

  return 0;
}
```

```
#define R(i) gpr(i)
#define Mr vaddr_read
#define Mw vaddr_write
```

Don’t Give Up Easy

- 我们既然知道 Makefile 里哪一行是 `.c → .o` 的转换
  - 我们添一个一模一样的 `gcc -E` 是不是就行了?

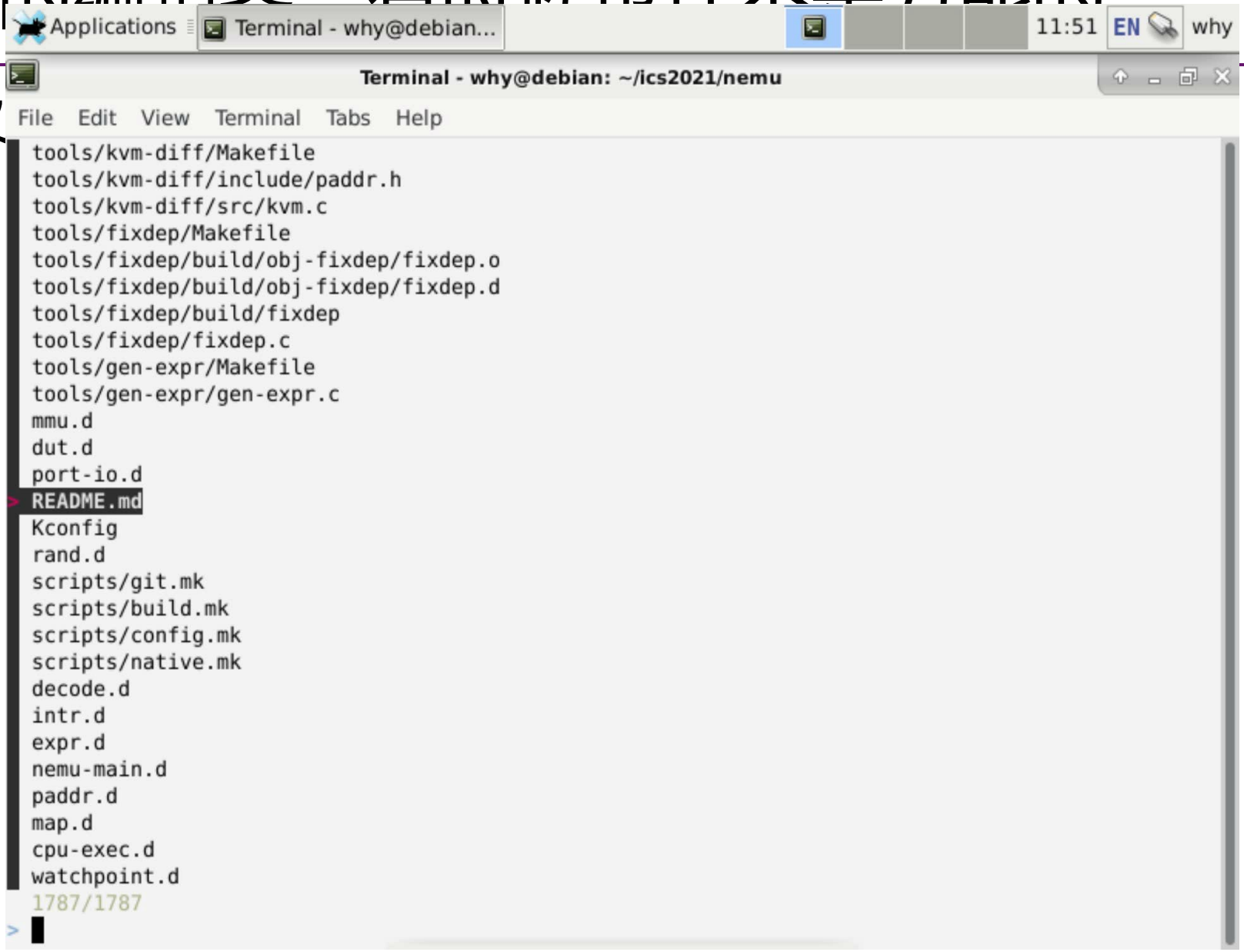
```
$(OBJ_DIR)/%.o: src/%.c
   @$(CC) $(CFLAGS) $(SO_CFLAGS) -c -o $@ $<
   @$(CC) $(CFLAGS) $(SO_CFLAGS) -E -MF /dev/null $< | \
       grep -ve '^#' | \
       clang-format - > $(basename $@).i
```

好的编辑器：有时候也许不是万能的

- 又爱又恨的元编程 （视频？）

好的编辑器：有时候也许不是万能的

- 又



Don’t Give Up Easy

- 我们既然知道 Makefile 里哪一行是 `.c → .o` 的转换
  - 我们添一个一模一样的 `gcc -E` 是不是就行了?

```
$(OBJ_DIR)/%.o: src/%.c
   @$(CC) $(CFLAGS) $(SO_CFLAGS) -c -o $@ $<
   @$(CC) $(CFLAGS) $(SO_CFLAGS) -E -MF /dev/null $< | \
       grep -ve '^#' | \
       clang-format - > $(basename $@).i
```

敲黑板：征服你畏惧的东西，就会有意想不到的收获。

总结

怎样读代码?

- 读代码 ≠ “读” 代码
  - 用正确的工具，使自己感到舒适
  - 但这个过程本身可能是不太舒适的 (走出你的舒适区)

  - 我们看到太多的同学，到最后都没有学会使用编辑器/IDE
    - 要相信一切不爽都有办法解决

- 信息来源
  - 在 /etc/hosts 中屏蔽百度
  - 去开源社区找 tutorials
    - 例子：vim-galore, awesome-c

End.

关于OJ

Ubuntu 20.04

Gcc 9.4.0

主要方式：观察Nemu控制台输出检查

recap：自我管理git

重装系统了？

虚拟机崩溃了？

环境配置出问题了？

PA2导读

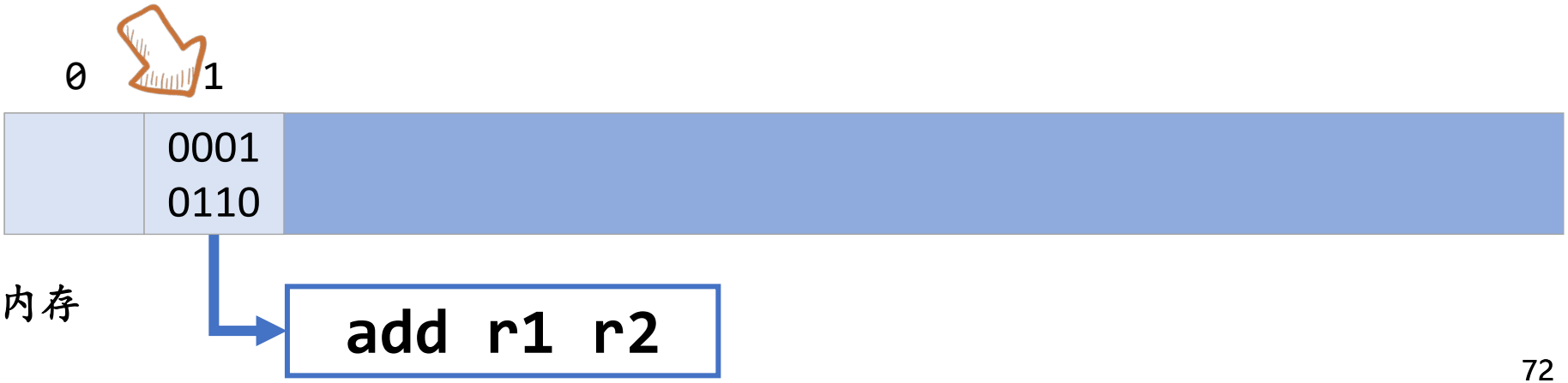
Lab1和PA2接踵而至,不要慌，慢慢来。

教科书第一章上的 “计算机系统”

存储系统and指令集

寄存器：PC，RO (RA)，R1，R2，R3 (8-bit)
内存：16字节（按字节访问）

```
                 7 6 5 4   3 2  1 0
mov            [0 0 0 0] [ rt][ rs]
add            [0 0 0 1] [ rt][ rs]
load           [1 1 1 0] [  addr  ]
store          [1 1 1 1] [  addr  ]
```



```c
#include <stdint.h>
#include <stdio.h>

#define NREG 4
#define NMEM 16

// 定义指令格式
typedef union {
  struct { uint8_t rs : 2, rt : 2, op : 4; } rtype;
  struct { uint8_t addr : 4      , op : 4; } mtype;
  uint8_t inst;
} inst_t;

#define DECODE_R(inst) uint8_t rt = (inst).rtype.rt, rs = (inst).rtype.rs
#define DECODE_M(inst) uint8_t addr = (inst).mtype.addr

uint8_t pc = 0;       // PC, C语言中没有4位的数据类型, 我们采用8位类型来表示
uint8_t R[NREG] = {}; // 寄存器
uint8_t M[NMEM] = {   // 内存, 其中包含一个计算z = x + y的程序
  0b11100110,  // load  6#     | R[0] <- M[y]
  0b00000100,  // mov   r1, r0 | R[1] <- R[0]
  0b11100101,  // load  5#     | R[0] <- M[x]
  0b00010001,  // add   r0, r1 | R[0] <- R[0] + R[1]
  0b11110111,  // store 7#     | M[z] <- R[0]
  0b00010000,  // x = 16
  0b00100001,  // y = 33
  0b00000000,  // z = 0
};
```

```c
int halt = 0; // 结束标志

// 执行一条指令
void exec_once() {
  inst_t this;
  this.inst = M[pc]; // 取指
  switch (this.rtype.op) {
  //  操作码译码        操作数译码            执行
    case 0b0000: { DECODE_R(this); R[rt]   = R[rs];   break; }
    case 0b0001: { DECODE_R(this); R[rt]  += R[rs];   break; }
    case 0b1110: { DECODE_M(this); R[0]    = M[addr]; break; }
    case 0b1111: { DECODE_M(this); M[addr] = R[0];    break; }
    default:
      printf("Invalid instruction with opcode = %x, halting...\n", this.rtype.op);
      halt = 1;
      break;
  }
  pc ++; // 更新PC
}

int main() {
  while (1) {
    exec_once();
    if (halt) break;
  }
  printf("The result of 16 + 33 is %d\n", M[7]);
  return 0;
}
```