CS 3502
Operating Systems

Project 1

Kun Suo

Computer Science, Kennesaw State University

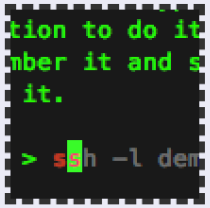
https://kevinsuo.github.io/

Command: fish

- https://fishshell.com/

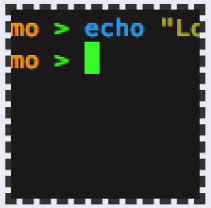
Autosuggestions

fish suggests commands as you type based on history and completions, just like a web browser. Watch out, Netscape Navigator 4.0!



Glorious VGA Color

fish supports 24 bit true color, the state of the art in terminal technology. Behold the monospaced rainbow.



Sane Scripting

fish is fully scriptable, and its syntax is simple, clean, and consistent. You'll never write `esac` again.

Web Based configuration

For those lucky few with a graphical computer, you can set your colors and view functions, variables, and history all from a web page.

Man Page Completions

Other shells support programmable completions, but only fish generates them automatically by parsing your installed man pages.

Works Out Of The Box

fish will delight you with features like tab completions and syntax highlighting that just work, with nothing new to learn or configure.

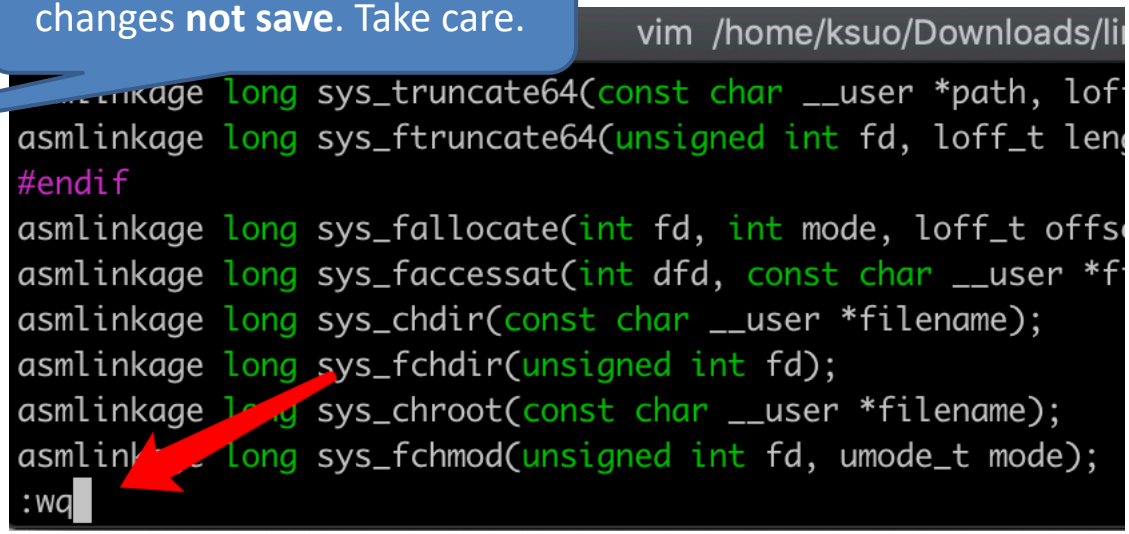
Project 1: No IDE, please use vim

- Open a file
  - $ vim test.c

- Close a file
  - Inside vim (after opening), press :q!
  - Inside vim (after opening), press :wq, changes saved

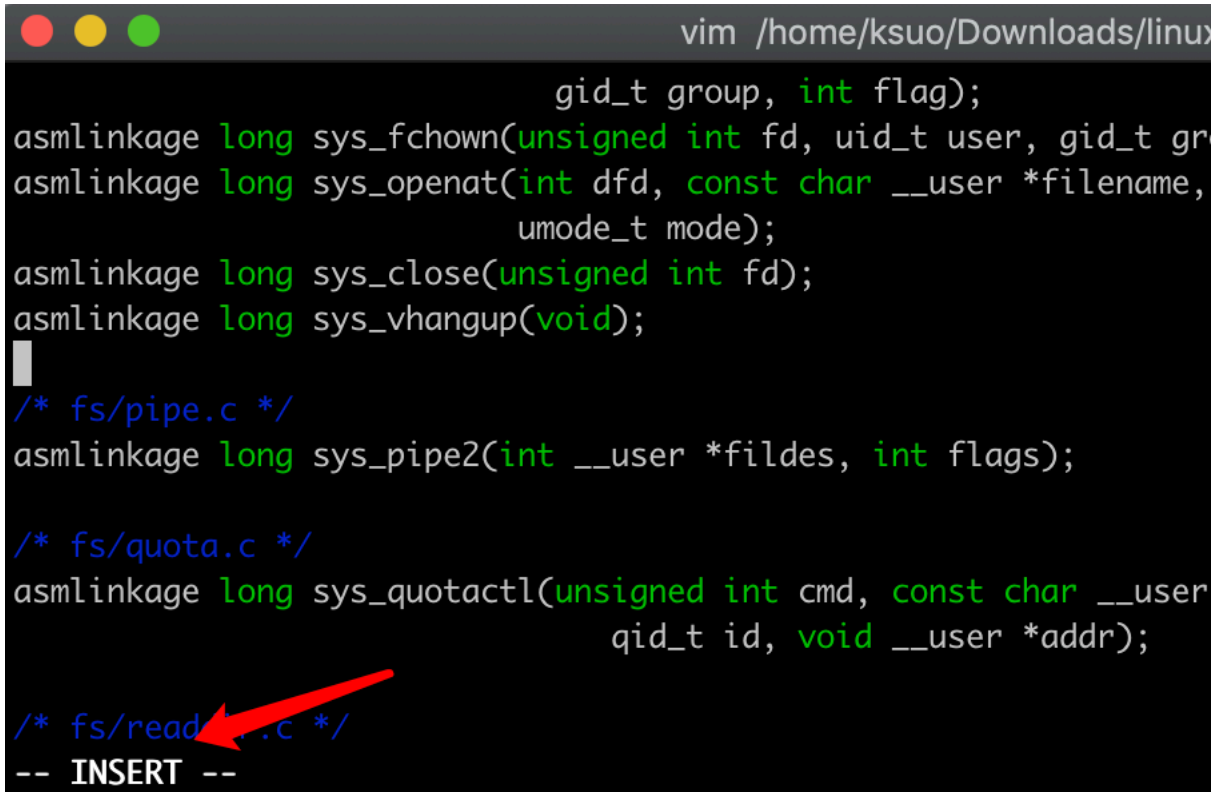
colon mark means menu.

Exclamation mark means changes **not save**. Take care.



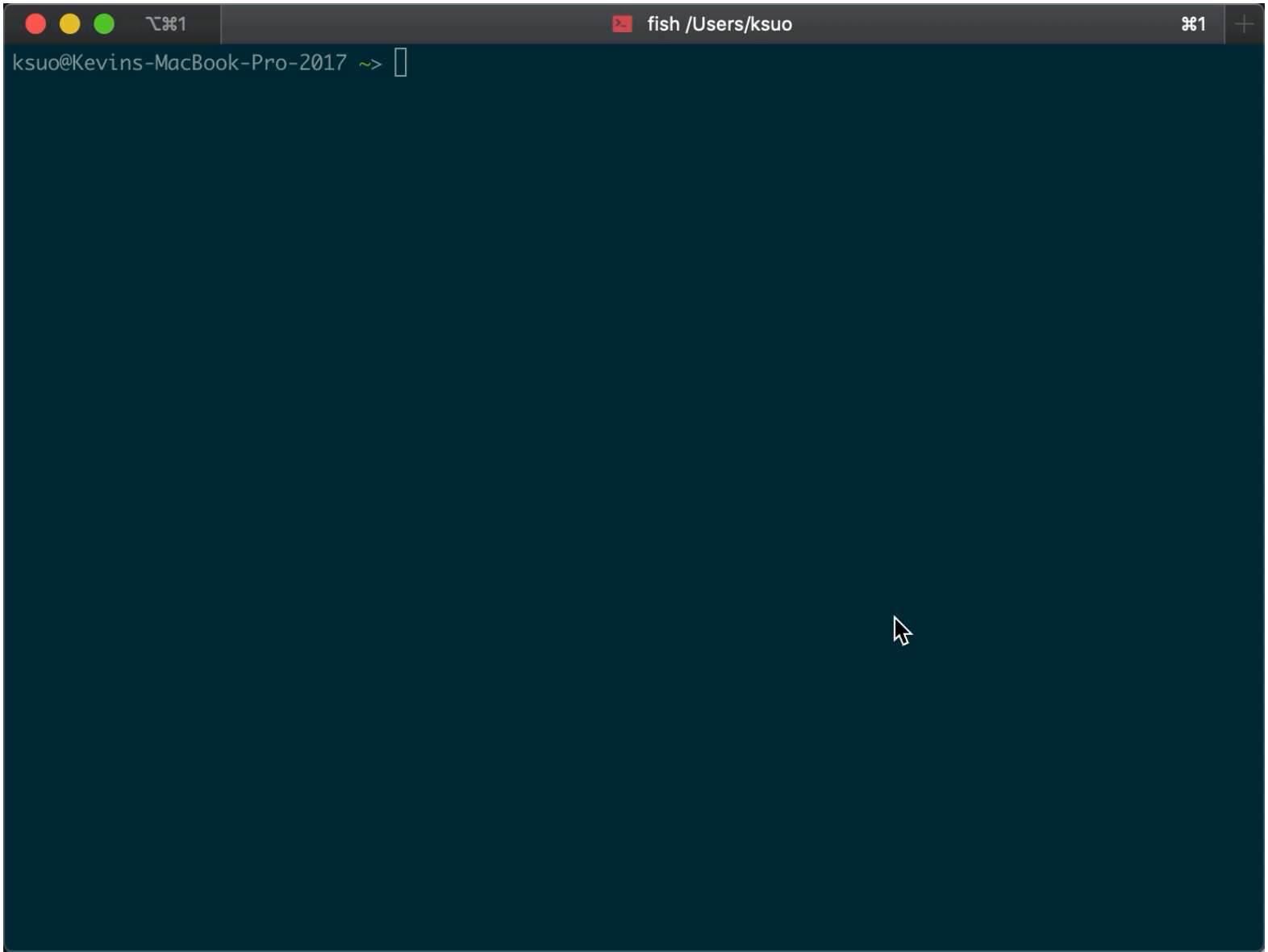
Project 1: No IDE, please use vim

- Edit a file
  - Open a file
  - Press i (means enter the insert mode). then input your code
  - Press ESC to exit the insert mode
  - Close a file (:wq)



Project 1: No IDE, please use vim

https://youtu.be/rS0XZpLtVlo



Edit a file with vim

- step 1: $ vim file
- step 2: press i, enter insert mode; move the cursor to position and edit the context
- step 3: after editing, press ESC to exit the insert mode to normal mode
- step 4: press :wq to save what you edit and quit. If you do not want to save, press :q!

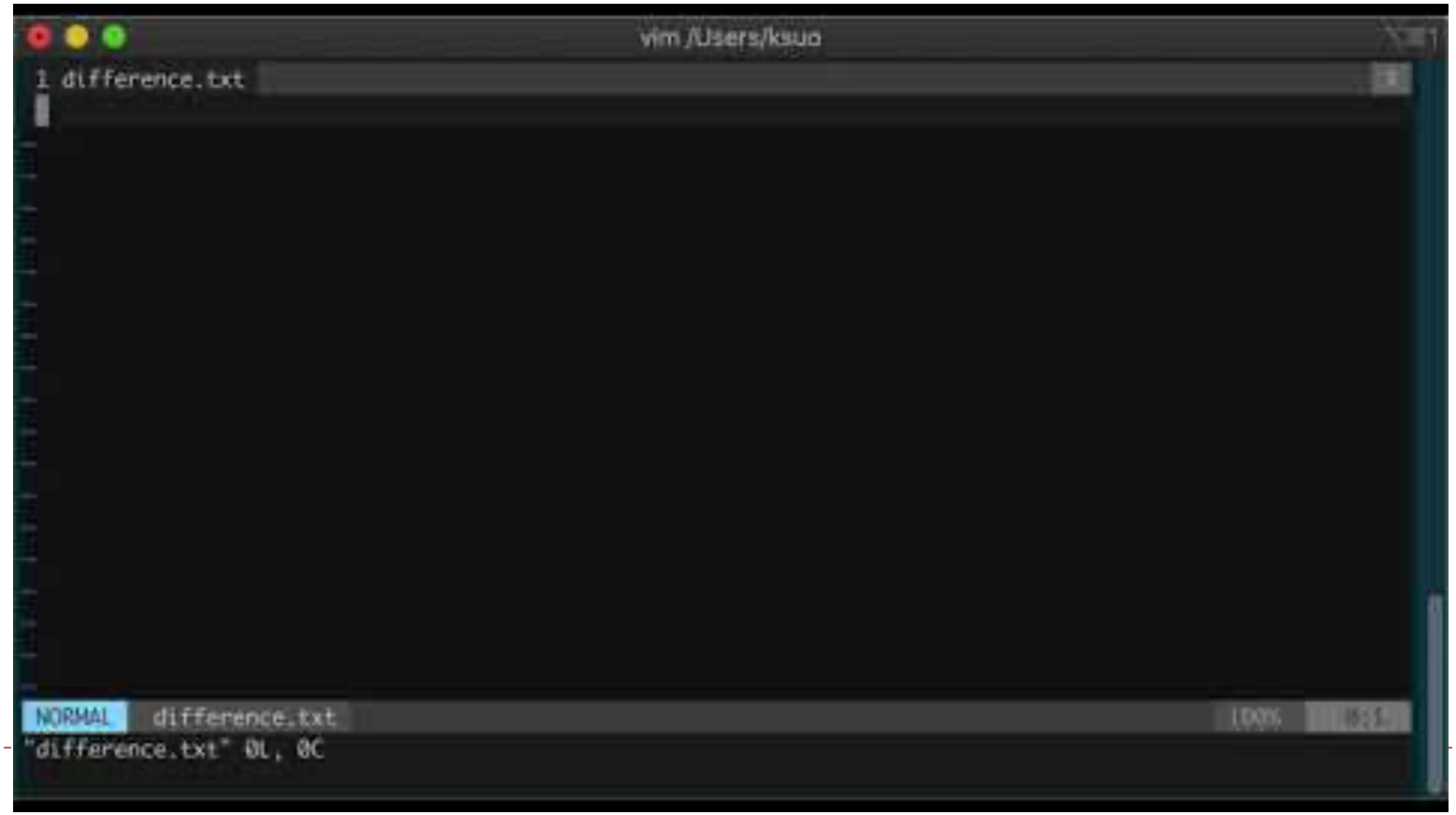
More about vim

- A quick start guide for beginners to the Vim text editor
  - https://eastmanreference.com/a-quick-start-guide-for-beginners-to-the-vim-text-editor
- Vim basics:
  - https://www.howtoforge.com/vim-basics
- Learn the Basic Vim Commands [Beginners Guide]
  - https://www.youtube.com/watch?time_continue=265&v=ZEGqkam-3Ic

Command: diff

https://youtu.be/oj0zJzJmdSs

- show the difference between two files:
  - $ diff -u original_file.c modified_file.c > result.txt

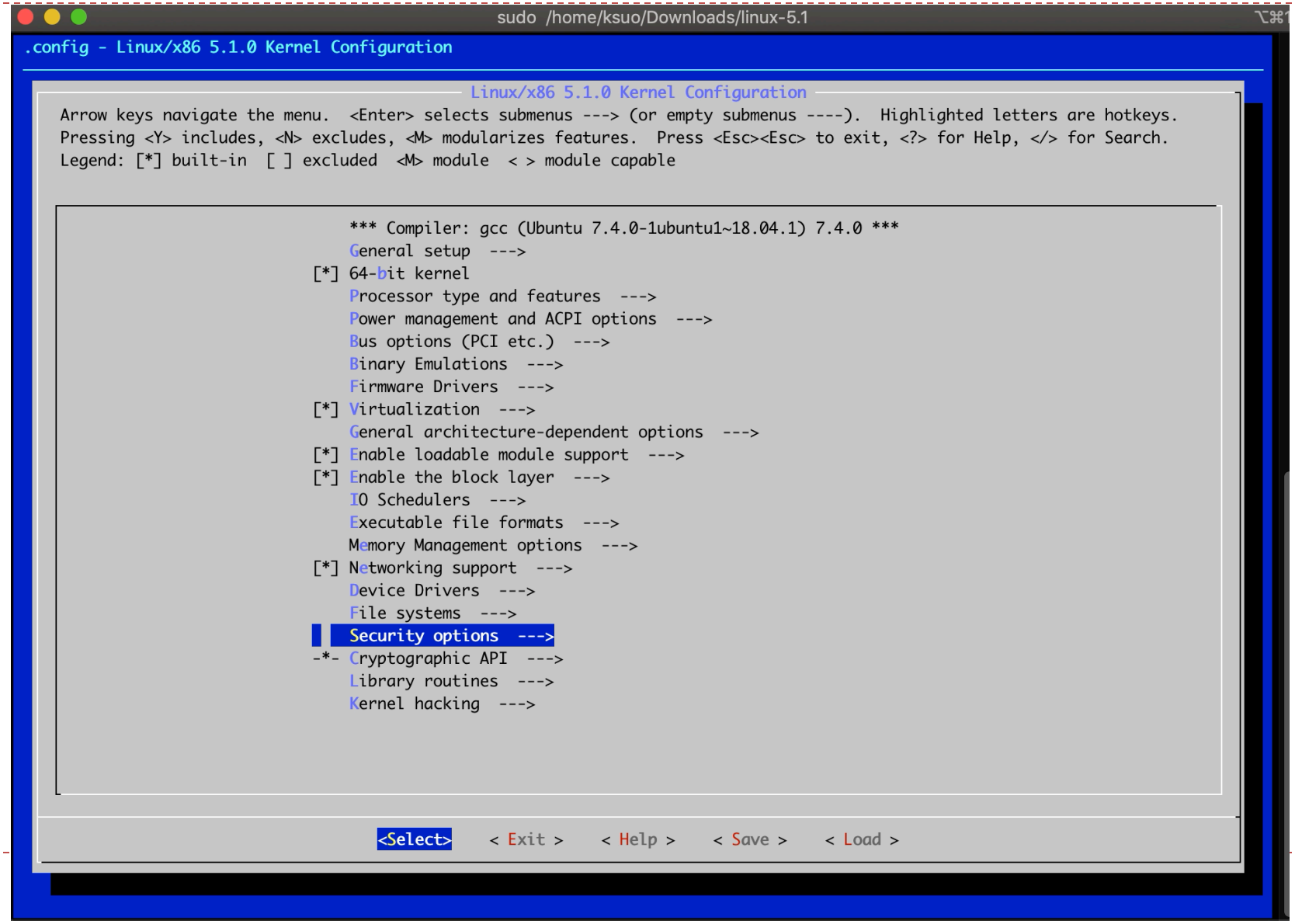


How to build one ubuntu VM?

- HostOS
  - Windows 10: https://www.youtube.com/watch?v=QbmRXJJKsvs
  - MacOS: https://www.youtube.com/watch?v=GDoCrfPma2k&t=321s

Project 1: menuconfig

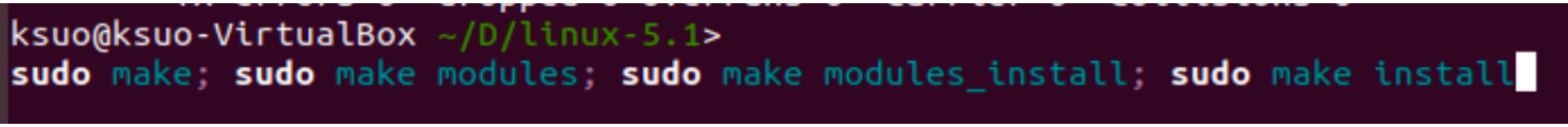
https://youtu.be/UyOGF4UOoR0



Tips

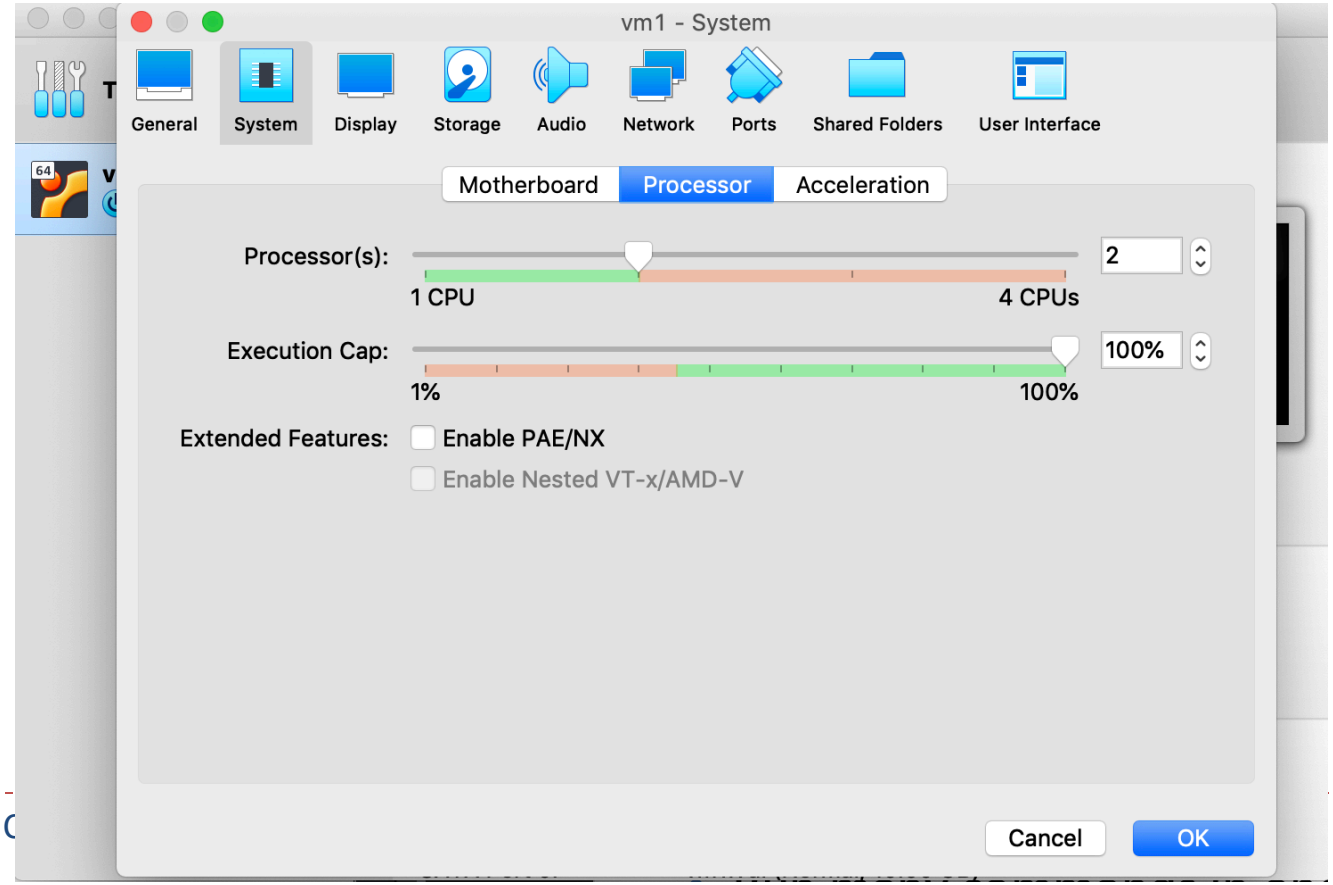
- Compile Linux kernel takes half to many hours, depending on machine speed
- To save the time, you can compile it before you sleep
- Run many commands in one:

  $ sudo make; sudo make modules; sudo make modules_install; sudo make install



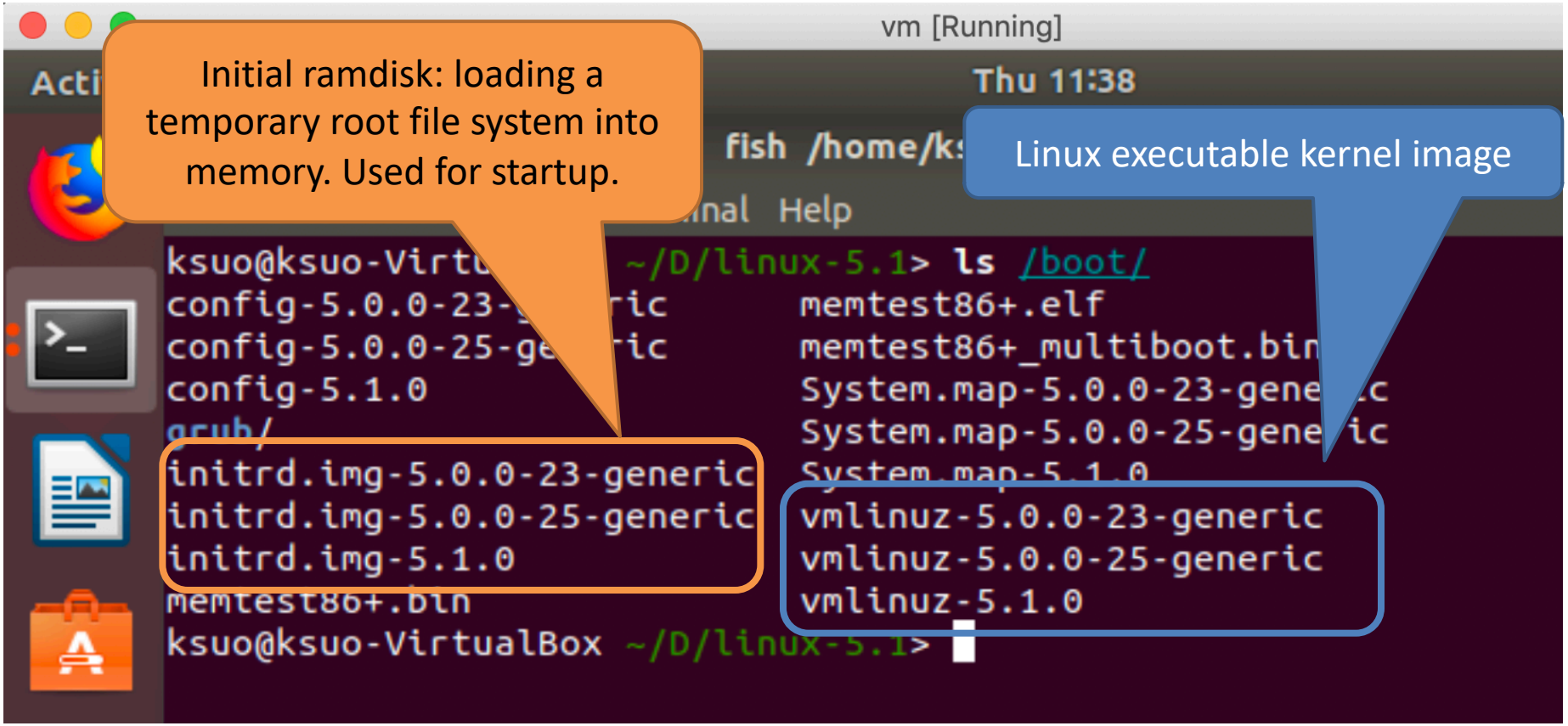
Tips

- Set up more CPUs for VM
- Make –j N, to accelerate compiling



Where is my kernel?

- $ ls /boot/



Which kernel to boot if there are many?

If you are using Ubuntu: change the grub configuration file:

$ sudo vim /etc/default/grub

Make the following changes:

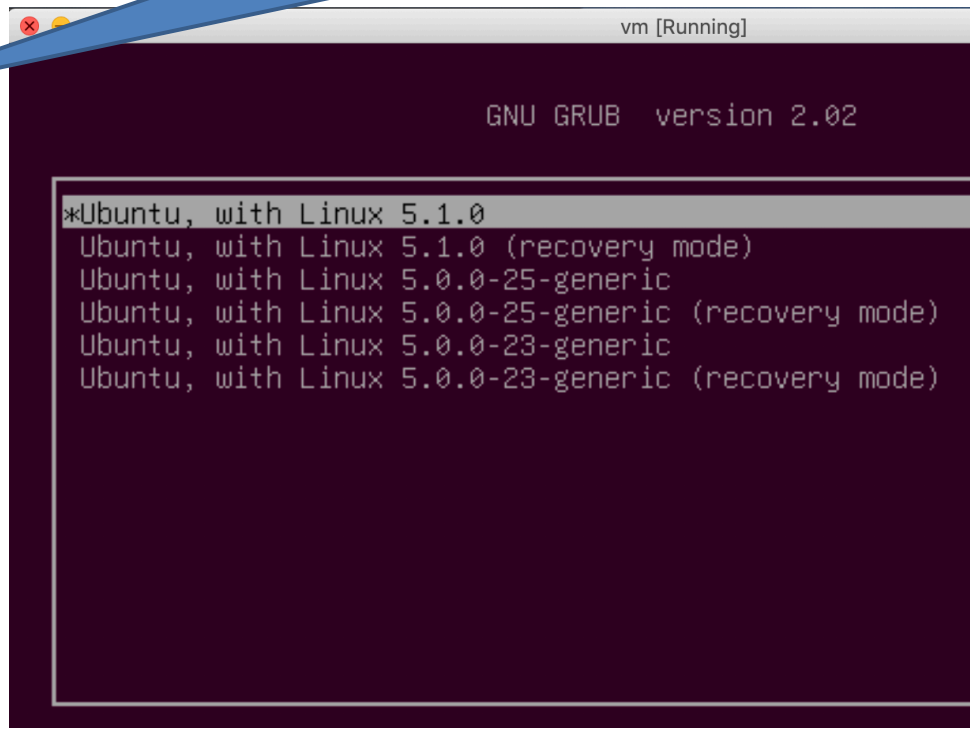
GRUB_DEFAULT=0

GRUB_TIMEOUT=10

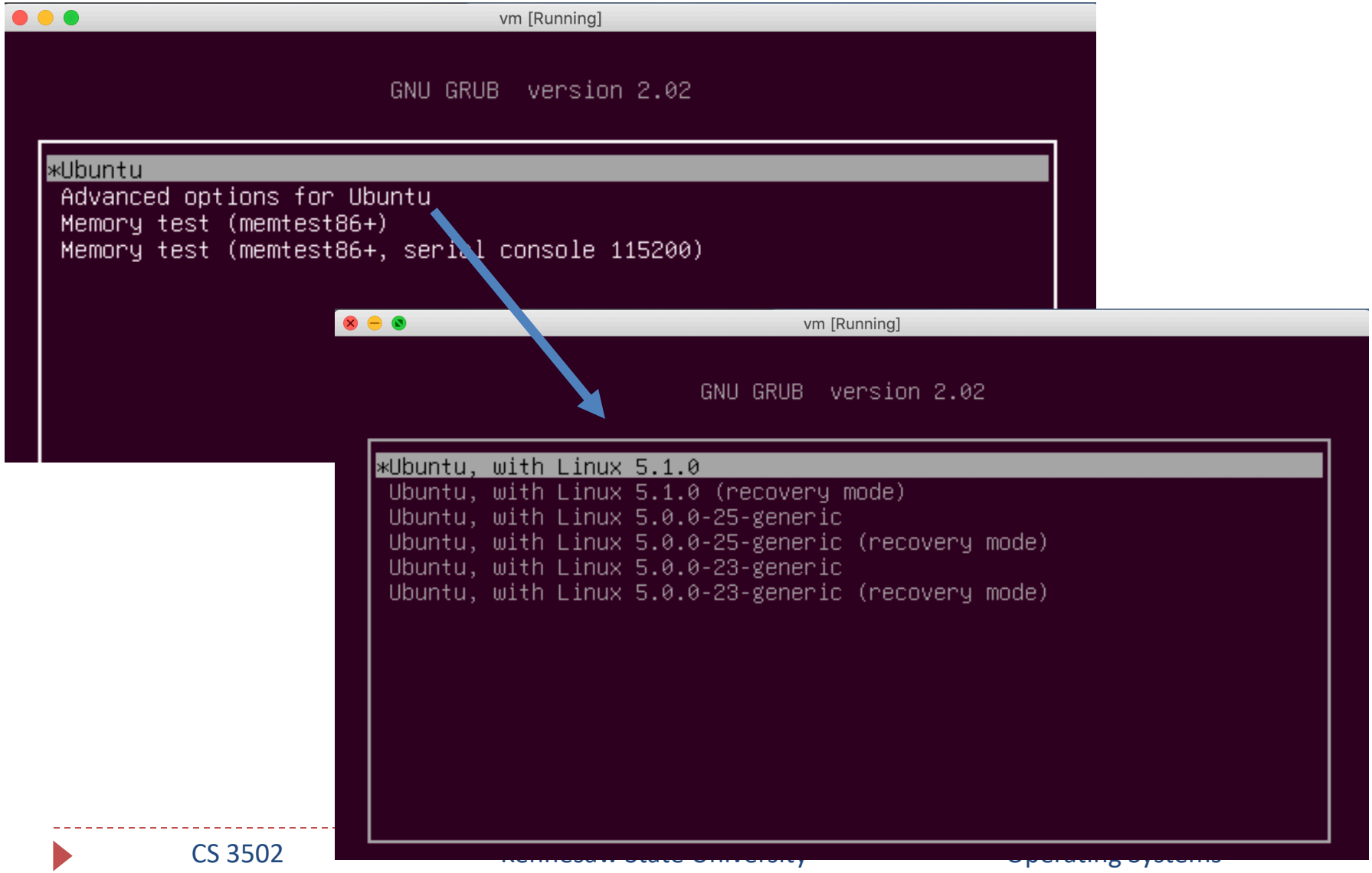
Then, update the grub entry:

$ sudo update-grub2

The OS boots by using the first kernel by default. You have 10 seconds to choose.



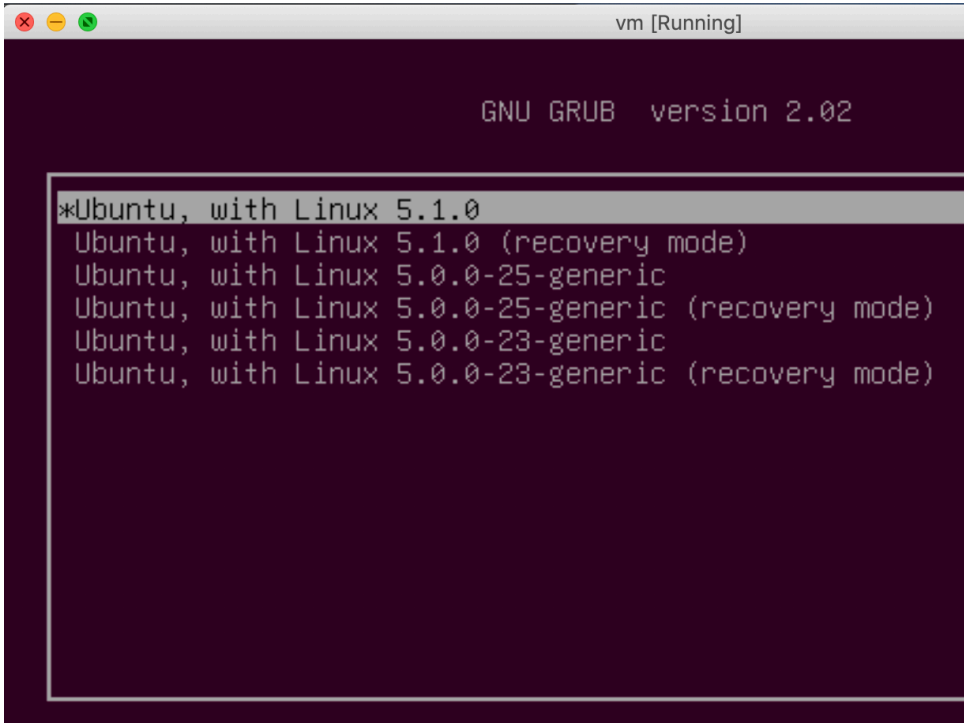
Which kernel to boot if there are many?



What if my kernel crashed?

- Your kernel could crash because you might bring in some kernel bugs

- In the menu, choose the old kernel to boot the system
- Fix your bug in the source code
- Compile and reboot



Compile the kernel

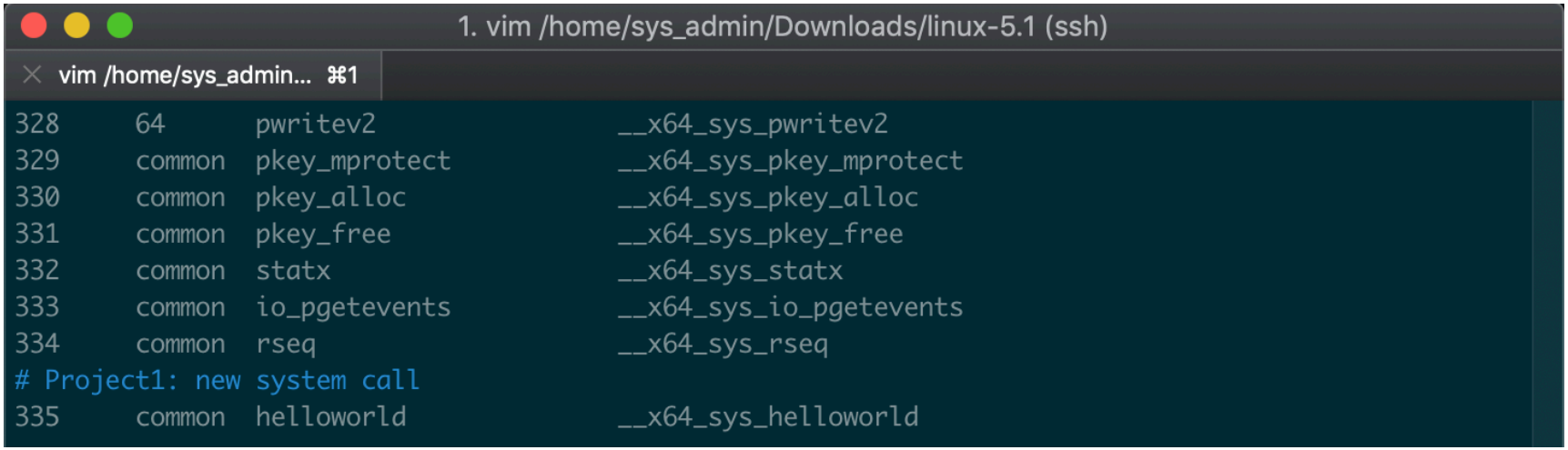
1. download the kernel source code
2. unzip the file
3. use *make menuconfig* to create a config file
4. use *make/make modules* to compile the code
5. use *make modules_install/make install* to install the kernel
6. reboot the VM

Define your system call

- Step 1: register your system call
- Step 2: declare your system call in the header file
- Step 3: implement your system call
- Step 4: write user level app to call it

Step 1: register your system call

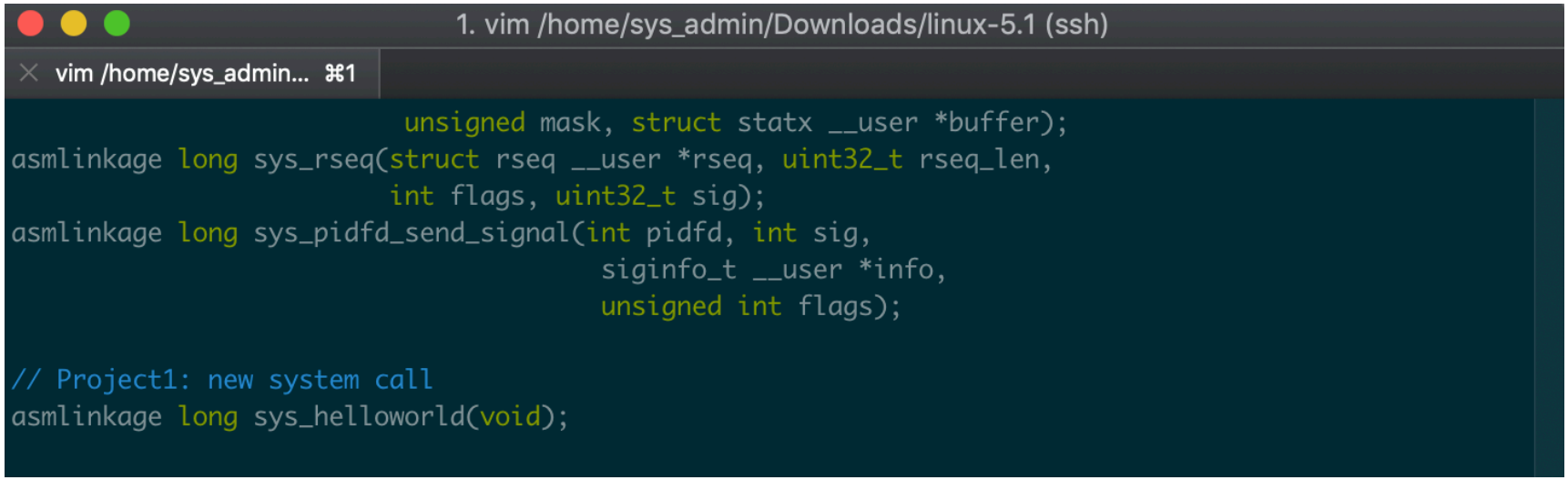
arch/x86/entry/syscalls/syscall_64.tbl



https://elixir.bootlin.com/linux/v5.0/source/arch/x86/entry/syscalls/syscall_64.tbl#L346

Step 2: declare your system call in the header file

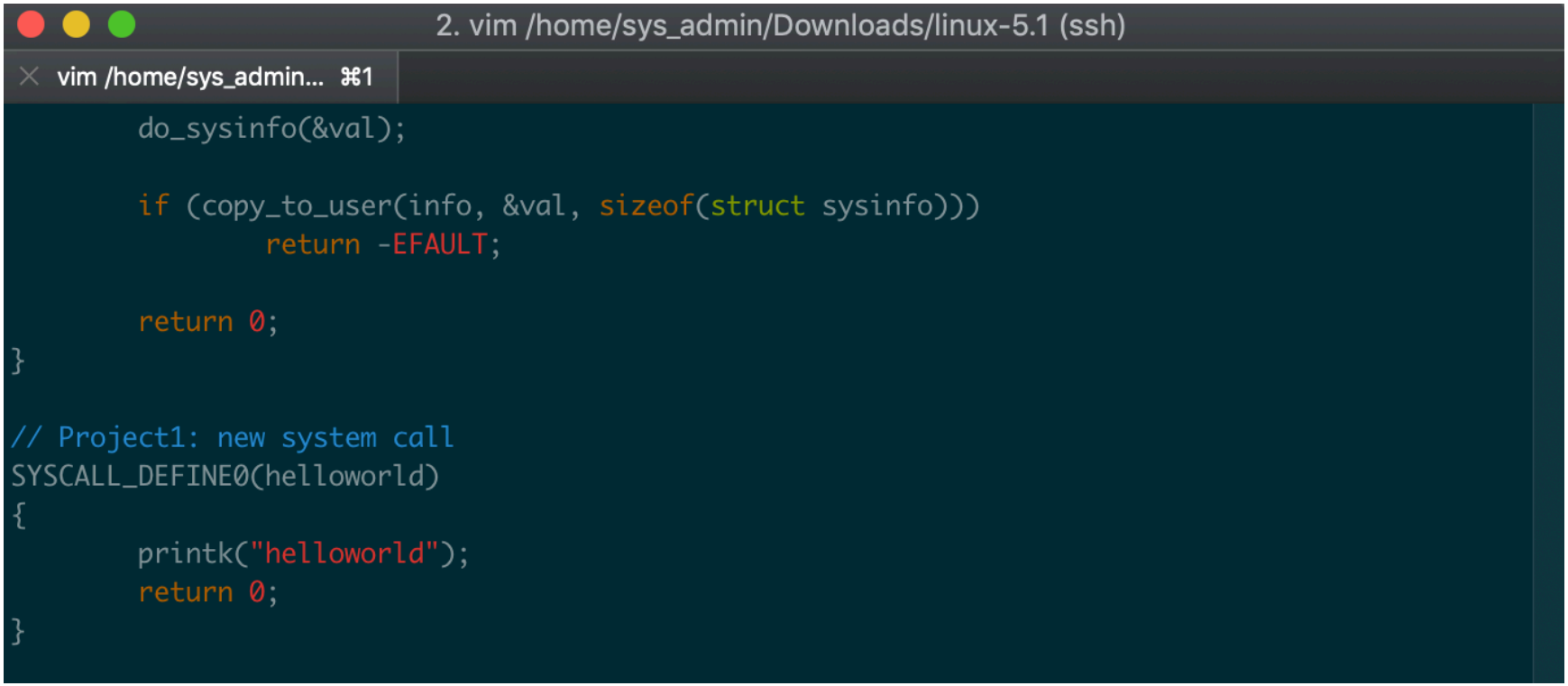
include/linux/syscalls.h



https://elixir.bootlin.com/linux/v5.0/source/include/linux/syscalls.h

Step 3: implement your system call

kernel/sys.c



https://elixir.bootlin.com/linux/v5.0/source/kernel/sys.c#L402

Step 4: write user level app to call it

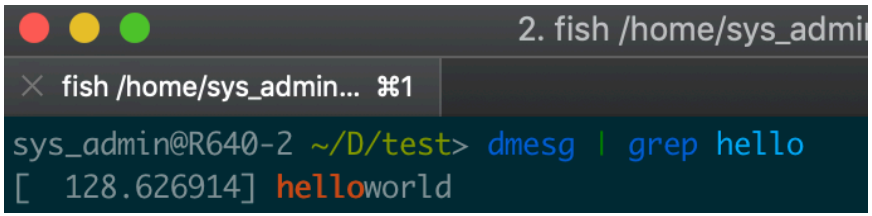
test_syscall.c：

```
****************************************
#include <linux/unistd .h>
#include <sys/syscall .h>
#include <sys/types .h>
#include <stdio .h>
#define __NR_helloworld 335

int main(int argc, char *argv[])
{
    syscall (__NR_helloworld) ;
    return 0 ;
}
****************************************

//If syscall needs parameter, then：
//syscall (__NR_helloworld，a，b，c) ;
```

Compile and execute：

```
$ gcc test_syscall.c -o test_syscall
$ ./test_syscall
```

The test program will call the new system call and output a helloworld message at the tail of the output of dmesg (system log).



Put it all together

User space

Kernel space

Step 4: user call it

test_syscall.c:

```
*****************************************
#include <linux/unistd .h>
#include <sys/syscall .h>
#include <sys/types .h>
#include <stdio .h>
#define __NR_helloworld 335

int main(int argc, char *argv[])
{
    syscall (__NR_helloworld) ;
    return 0 ;
}
*****************************************
```

Step 1: register

1. vim /home/sys_admin/Downloads/linux-5.1 (

vim /home/sys_admin... ⌘1

```
328     64      pwritev2                __x64_sys_pwritev2
329     common  pkey_mprotect           __x64_sys_pkey_mprotect
330     common  pkey_alloc              __x64_sys_pkey_alloc
331     common  pkey_free               __x64_sys_pkey_free
332     common  statx                   __x64_sys_statx
333     common  io_pgetevents           __x64_sys_io_pgetevents
334     common  rseq                    __x64_sys_rseq
# Project1: new system call
335     common  helloworld              __x64_sys_helloworld
```

Step 2: declare

1. vim /home/sys_admin/Downloads/linux-5.1 (ssh

vim /home/sys_admin... ⌘1

```
                        unsigned mask, struct statx __user *buffer);
asmlinkage long sys_rseq(struct rseq __user *rseq, uint32_t rseq_len,
                         int flags, uint32_t sig);
asmlinkage long sys_pidfd_send_signal(int pidfd, int sig,
                                      siginfo_t __user *info,
                                      unsigned int flags);

// Project1: new system call
asmlinkage long sys_helloworld(void);
```

2. vim /home/sys_admin/Downloads/linux

vim /home/sys_admin... ⌘1

```
        do_sysinfo(&val);

        if (copy_to_user(info, &val, sizeof(struct sysinfo)))
                return -EFAULT;

        return 0;
}

// Project1: new system call
SYSCALL_DEFINE0(helloworld)
{
        printk("helloworld");
        return 0;
}
```

Step 3: implementation

2. fish /home/sys_admi

fish /home/sys_admin... ⌘1

```
sys_admin@R640-2 ~/D/test> dmesg | grep hello
[  128.626914] helloworld
```