CS 3502
Operating Systems

Project 1

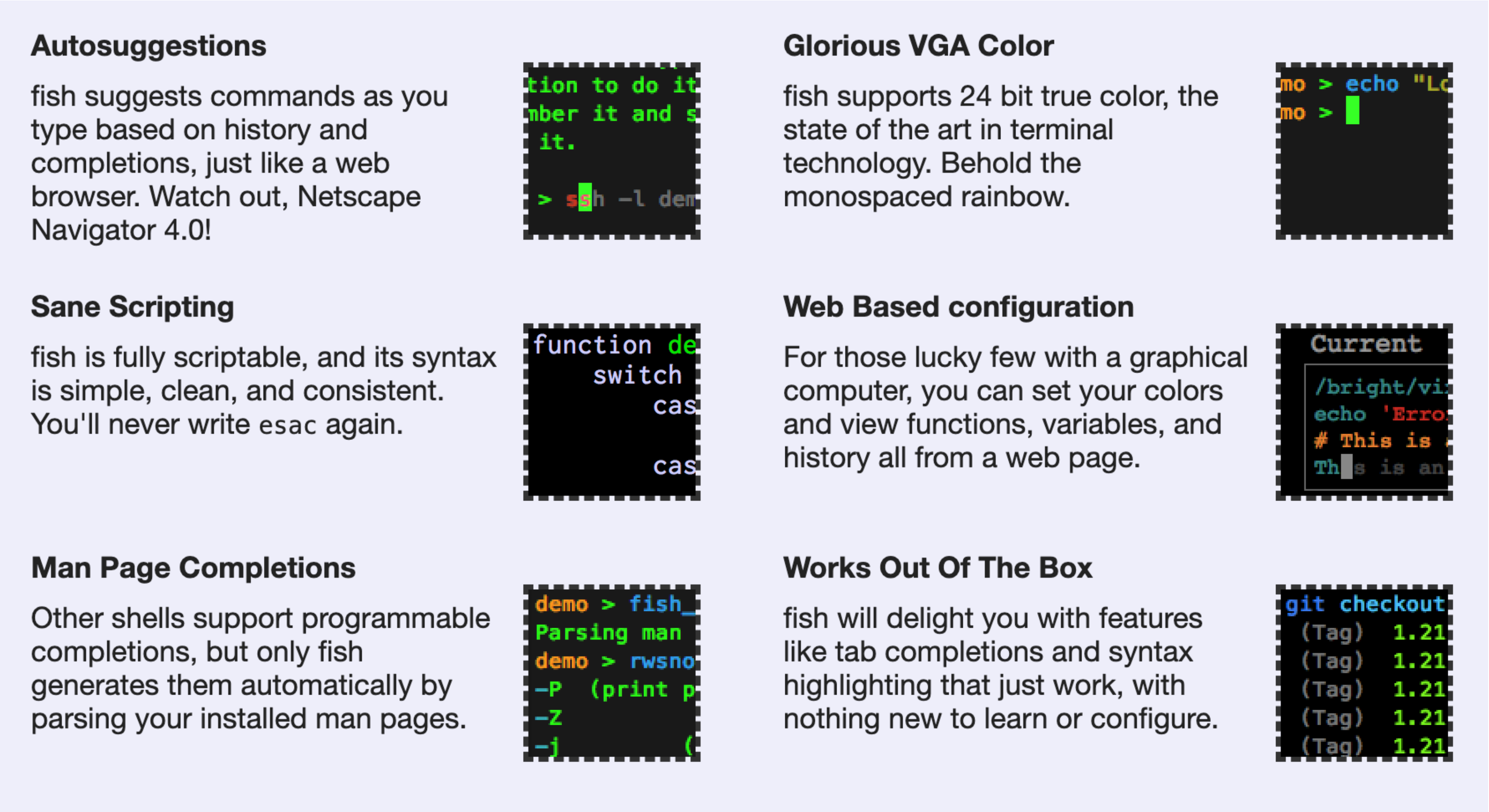
Kun Suo

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!

```
tion to do it
nber it and s
 it.

> ssh -l dem
```

Glorious VGA Color

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

```
mo > echo "Lc
mo >
```

Sane Scripting

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

```
function de
    switch
        cas

        cas
```

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.

```
 Current
 /bright/vi
 echo 'Erro
 # This is
 Th s is an
```

Man Page Completions

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

```
demo > fish_
Parsing man
demo > rwsno
-P  (print p
-Z
-j         (
```

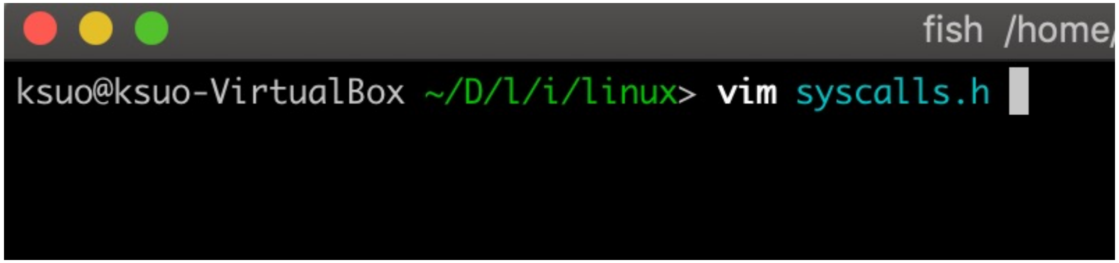
Works Out Of The Box

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

```
git checkout
 (Tag)  1.21
 (Tag)  1.21
 (Tag)  1.21
 (Tag)  1.21
 (Tag)  1.21
```

Project 1: No IDE, please use vim

- Open a file
  - $ vim test.c

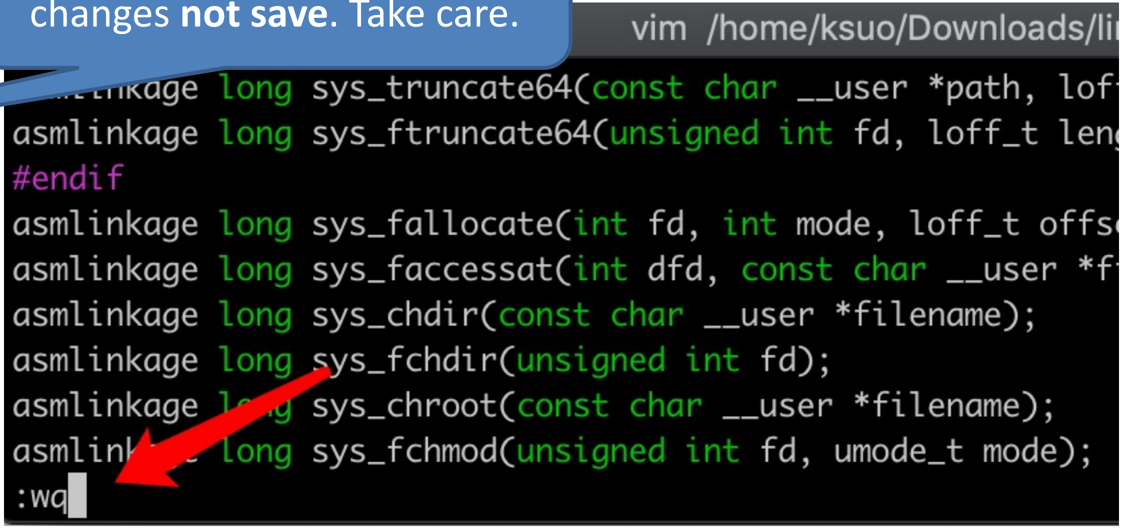


```
ksuo@ksuo-VirtualBox ~/D/l/i/linux> vim syscalls.h
```

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

colon mark means menu.

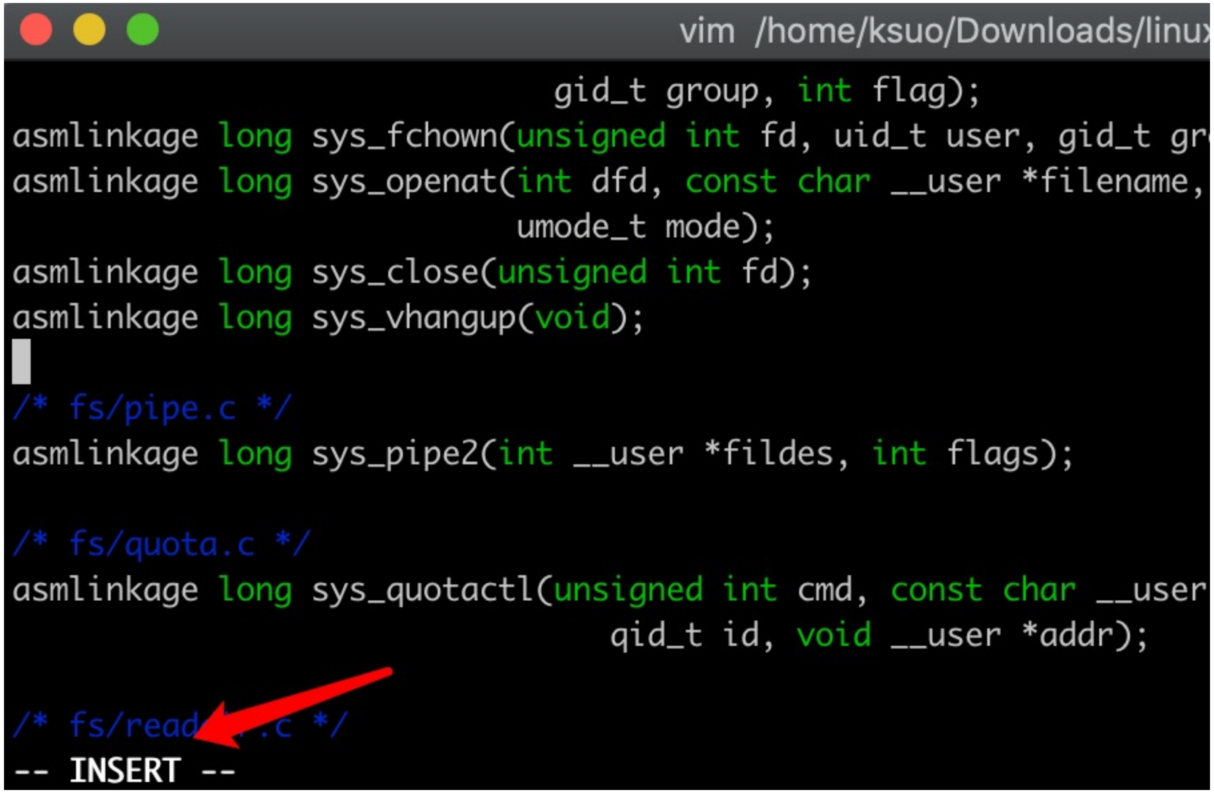
Exclamation mark means changes **not save**. Take care.



```
asmlinkage long sys_truncate64(const char __user *path, lof
asmlinkage long sys_ftruncate64(unsigned int fd, loff_t len
#endif
asmlinkage long sys_fallocate(int fd, int mode, loff_t offs
asmlinkage long sys_faccessat(int dfd, const char __user *f
asmlinkage long sys_chdir(const char __user *filename);
asmlinkage long sys_fchdir(unsigned int fd);
asmlinkage long sys_chroot(const char __user *filename);
asmlinkage long sys_fchmod(unsigned int fd, umode_t mode);
:wq
```

Project 1: No IDE, please use vim

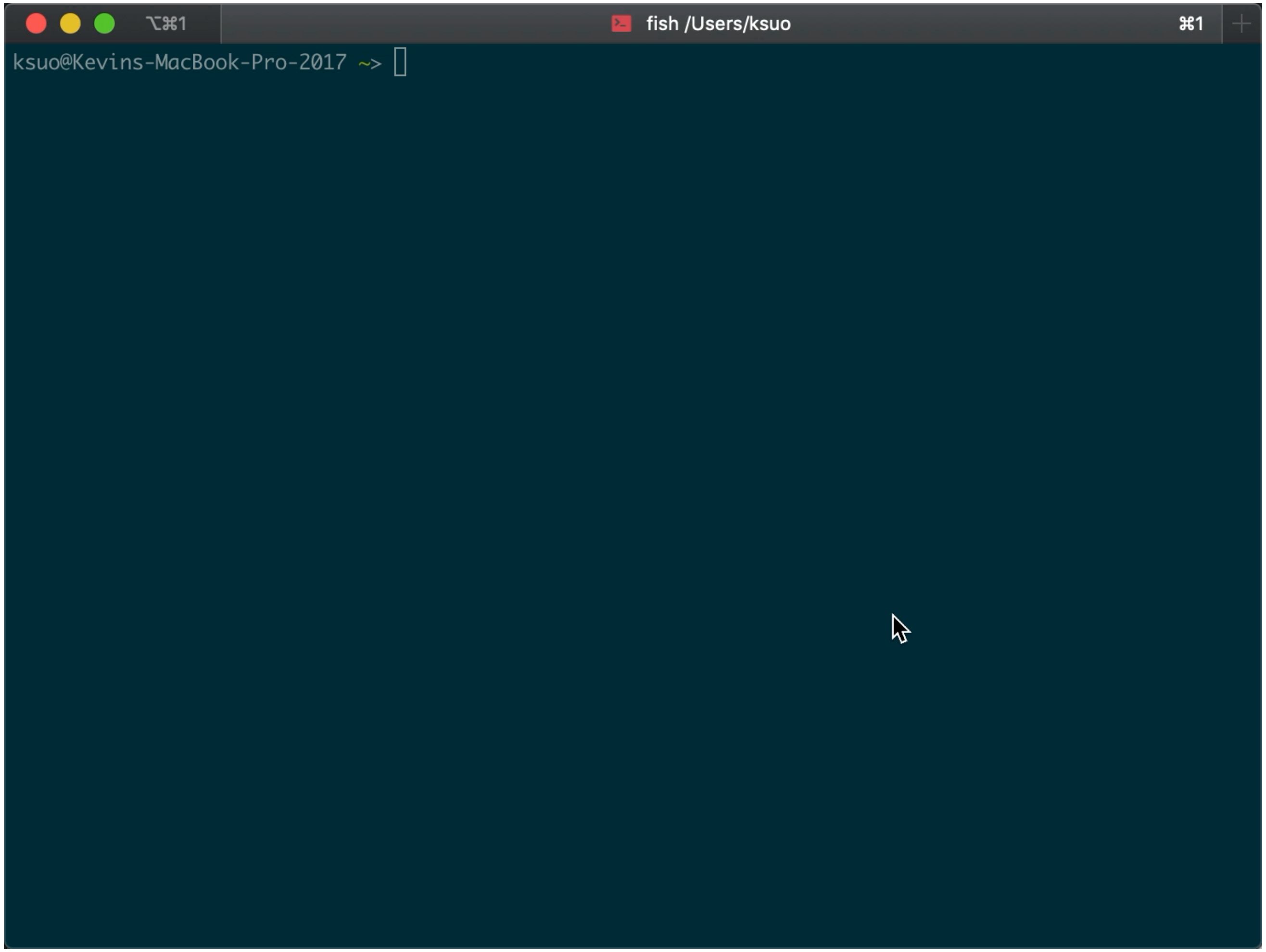
- Edit a file
  - Open a file
  - Press i (means enter the insert mode). then input your code
  - Press ESC to exit the insert mode
  - Close a file (:wq)



```
                              gid_t group, int flag);
asmlinkage long sys_fchown(unsigned int fd, uid_t user, gid_t gr
asmlinkage long sys_openat(int dfd, const char __user *filename,
                             umode_t mode);
asmlinkage long sys_close(unsigned int fd);
asmlinkage long sys_vhangup(void);


/* fs/pipe.c */
asmlinkage long sys_pipe2(int __user *fildes, int flags);

/* fs/quota.c */
asmlinkage long sys_quotactl(unsigned int cmd, const char __user
                                  qid_t id, void __user *addr);


/* fs/read    .c */
-- INSERT --
```

Project 1: No IDE, please use vim

https://youtu.be/rS0XZpLtVlo



```
ksuo@Kevins-MacBook-Pro-2017 ~>
```

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

Outline

- **Part A: create a VM and compile your kernel**
- Part B: add a new system call into the Linux kernel

How to build one ubuntu VM?

- HostOS

Windows 10 (x86):

https://www.youtube.com/watch?v=QbmRXJJKsvs

MacOS (x86):

https://www.youtube.com/watch?v=GDoCrfPma2k&t=321s

MacOS (arm):

https://youtu.be/O19mv1pe76M?si=4cYayFiqPNoHoY1w

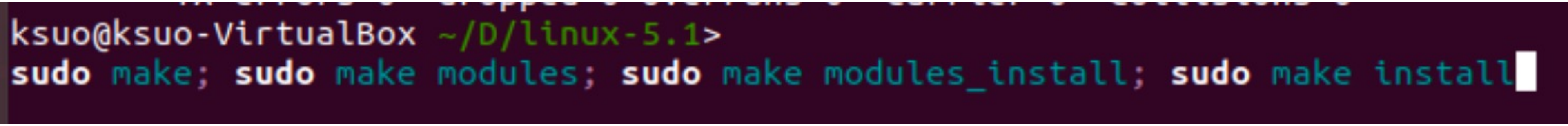
Tips

- Compile Linux kernel takes half to two hours, depending on machine speed

- To save the time, use $ make localmodconfig

- Run many commands in one:

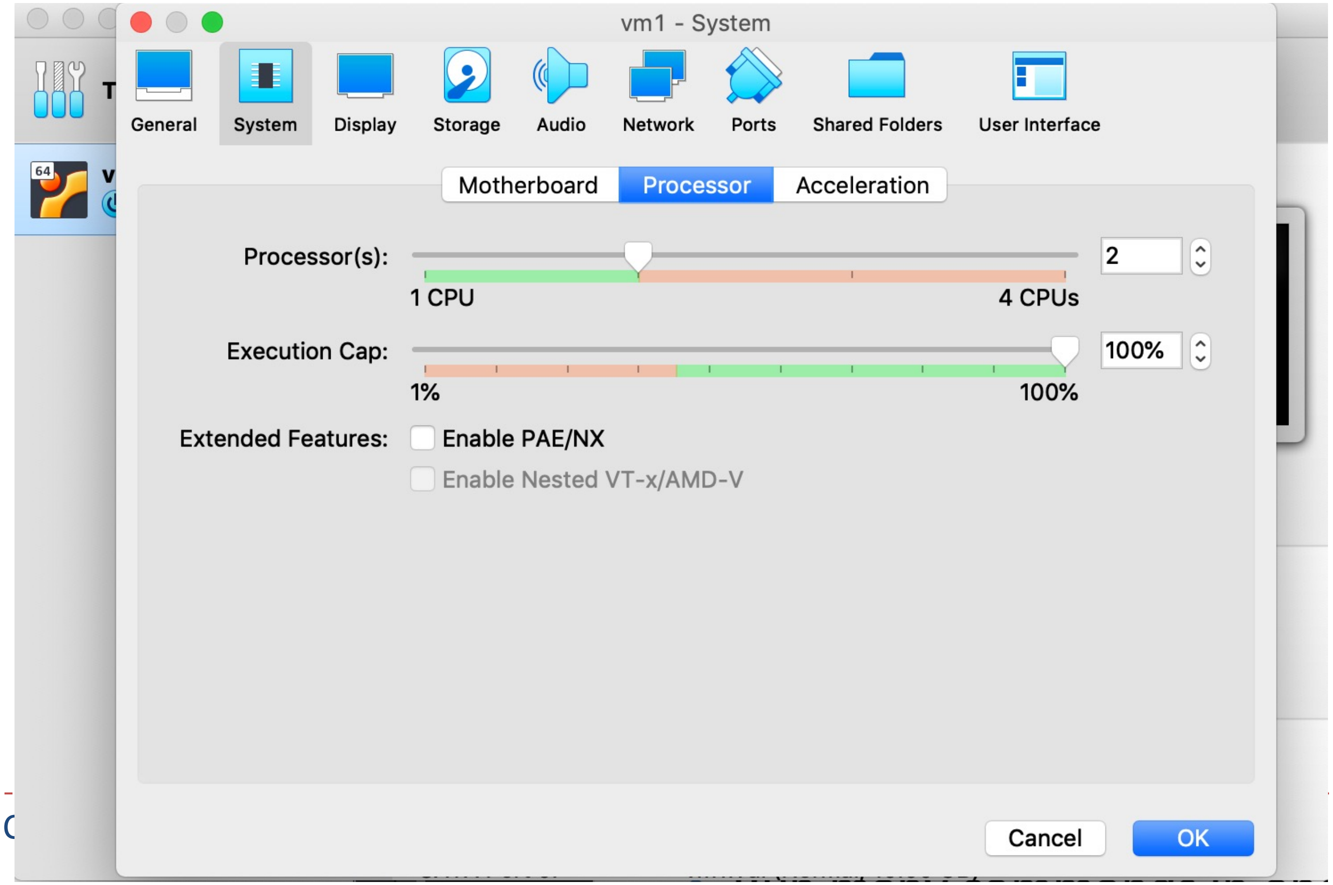
  $ sudo make; sudo make modules; sudo make modules_install; sudo make install



```
ksuo@ksuo-VirtualBox ~/D/linux-5.1>
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 (N is your CPU number)



Processor(s): 2 (1 CPU – 4 CPUs)

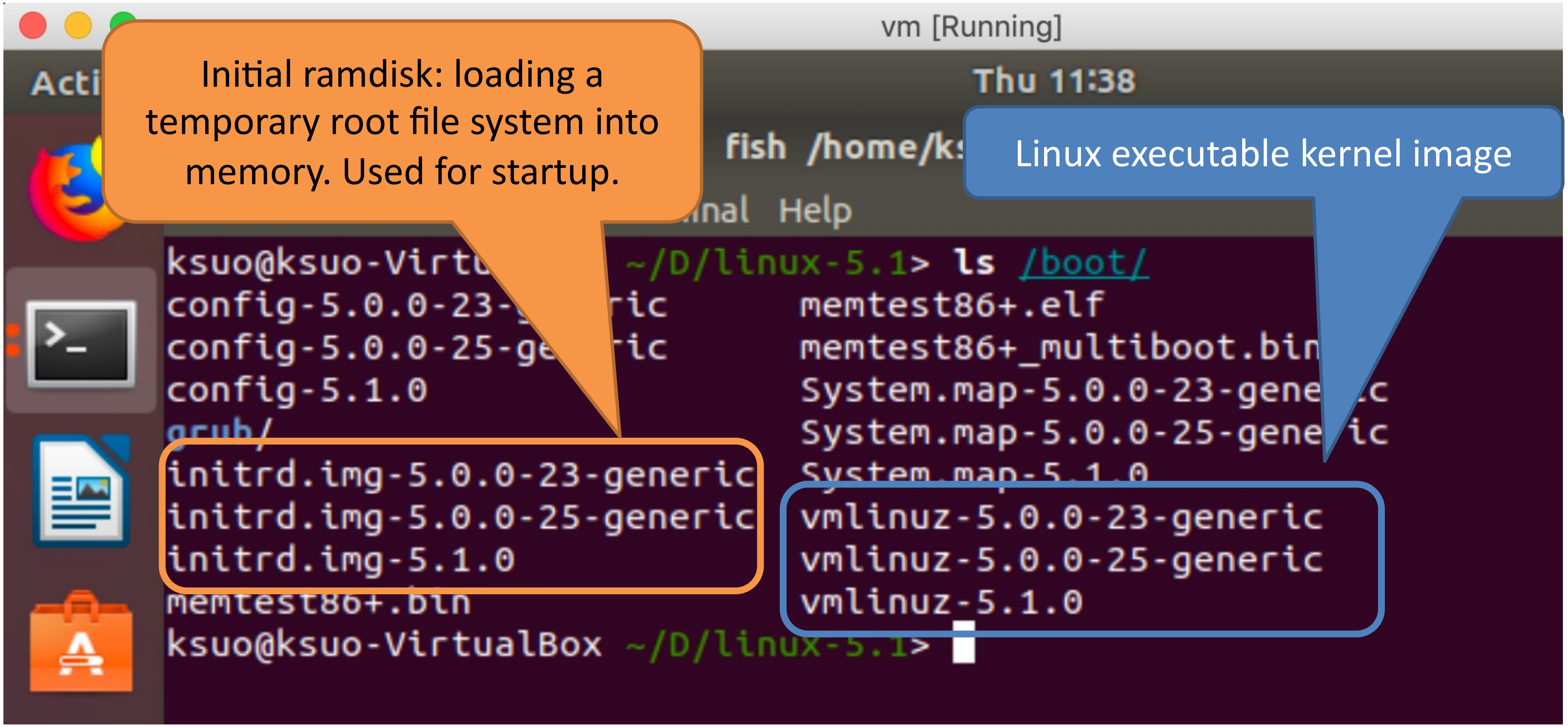
Execution Cap: 100% (1% – 100%)

Extended Features: □ Enable PAE/NX

□ Enable Nested VT-x/AMD-V

Where is my kernel?

- $ ls /boot/



```
ksuo@ksuo-Virtu         ~/D/linux-5.1> ls /boot/
config-5.0.0-23-     ric        memtest86+.elf
config-5.0.0-25-ge    ric       memtest86+_multiboot.bi
config-5.1.0                    System.map-5.0.0-23-gene  c
grub/                           System.map-5.0.0-25-gene  ic
initrd.img-5.0.0-23-generic     System.map-5.1.0
initrd.img-5.0.0-25-generic     vmlinuz-5.0.0-23-generic
initrd.img-5.1.0                vmlinuz-5.0.0-25-generic
memtest86+.bin                  vmlinuz-5.1.0
ksuo@ksuo-VirtualBox ~/D/linux-5.1>
```

Initial ramdisk: loading a temporary root file system into memory. Used for startup.

Linux executable kernel image

Which kernel to boot if there are many?

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

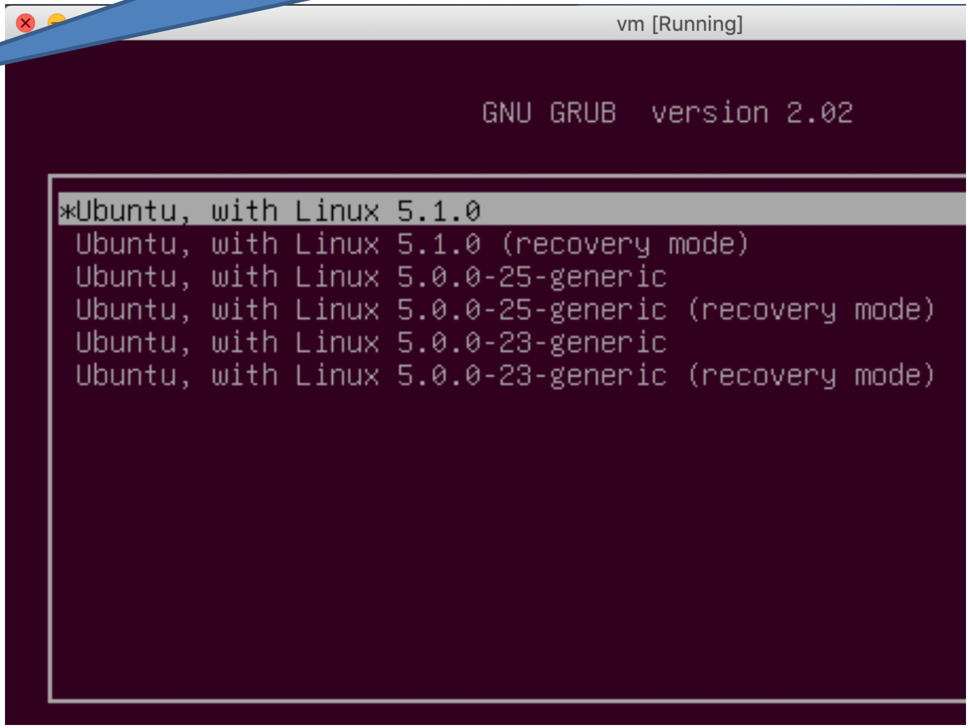
```
$ sudo vim /etc/default/grub
```

Make the following changes:

```
GRUB_DEFAULT=0
GRUB_TIMEOUT=10
```

Then, update the grub entry:

```
$ sudo update-grub2
```

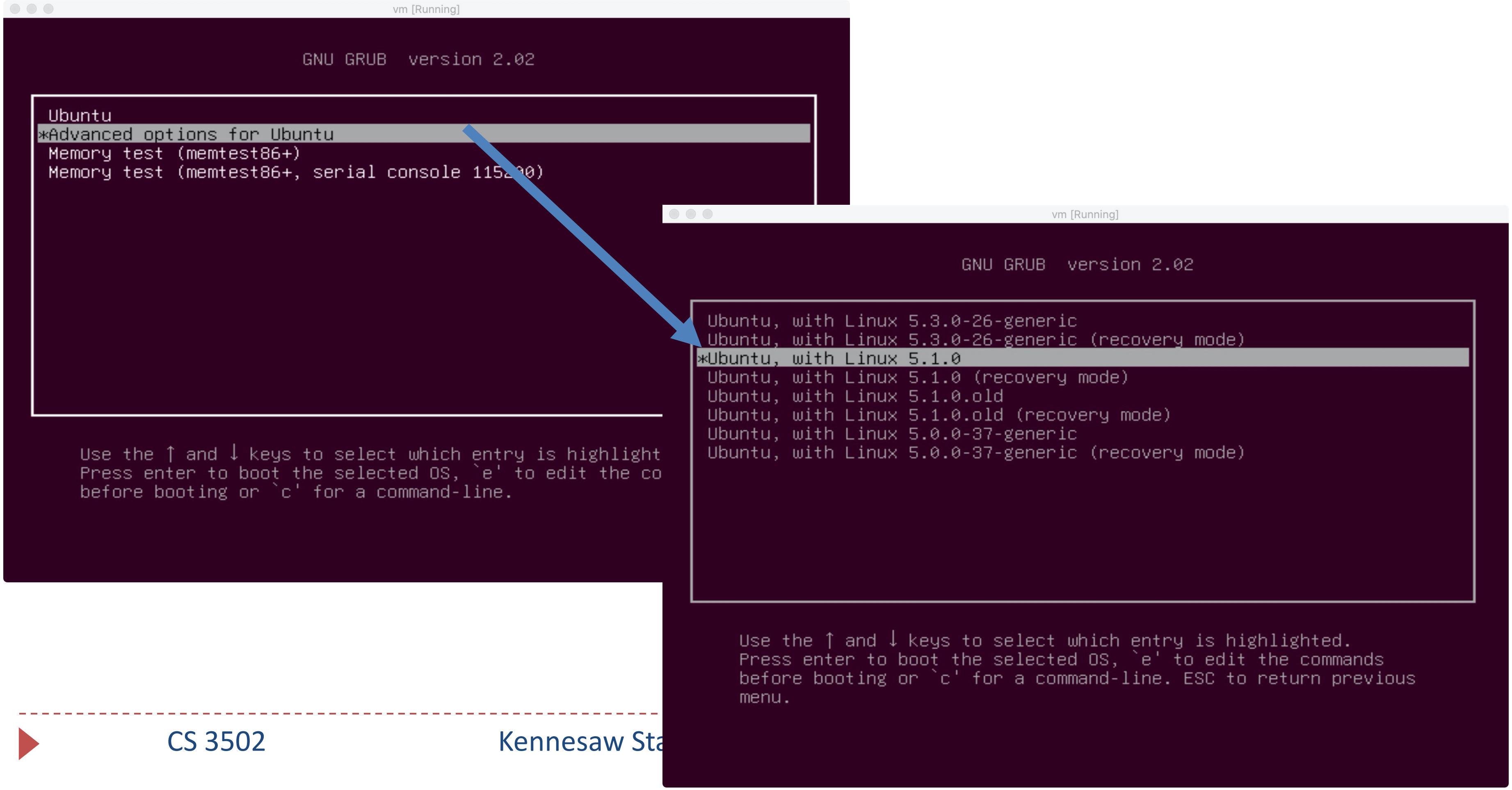


```
GNU GRUB  version 2.02

*Ubuntu, with Linux 5.1.0
 Ubuntu, with Linux 5.1.0 (recovery mode)
 Ubuntu, with Linux 5.0.0-25-generic
 Ubuntu, with Linux 5.0.0-25-generic (recovery mode)
 Ubuntu, with Linux 5.0.0-23-generic
 Ubuntu, with Linux 5.0.0-23-generic (recovery mode)
```

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

Which kernel to boot if there are many?

Immediately after the BIOS/UEFI splash screen during boot, with BIOS, quickly press and hold the ***Shift*** key, which will bring up the GNU GRUB menu. (If you see the Ubuntu logo, you've missed the point where you can enter the GRUB menu.)



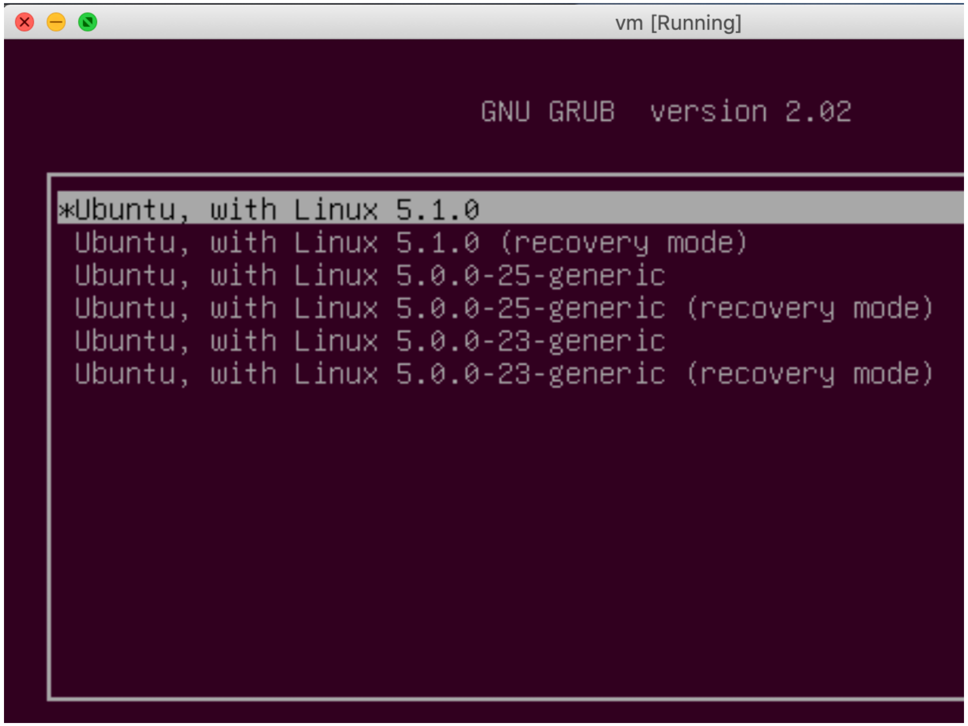
```
GNU GRUB  version 2.02

 Ubuntu
*Advanced options for Ubuntu
 Memory test (memtest86+)
 Memory test (memtest86+, serial console 115200)

   Use the ↑ and ↓ keys to select which entry is highlight
   Press enter to boot the selected OS, `e' to edit the co
   before booting or `c' for a command-line.
```

```
GNU GRUB  version 2.02

 Ubuntu, with Linux 5.3.0-26-generic
 Ubuntu, with Linux 5.3.0-26-generic (recovery mode)
*Ubuntu, with Linux 5.1.0
 Ubuntu, with Linux 5.1.0 (recovery mode)
 Ubuntu, with Linux 5.1.0.old
 Ubuntu, with Linux 5.1.0.old (recovery mode)
 Ubuntu, with Linux 5.0.0-37-generic
 Ubuntu, with Linux 5.0.0-37-generic (recovery mode)

   Use the ↑ and ↓ keys to select which entry is highlighted.
   Press enter to boot the selected OS, `e' to edit the commands
   before booting or `c' for a command-line. ESC to return previous
   menu.
```

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

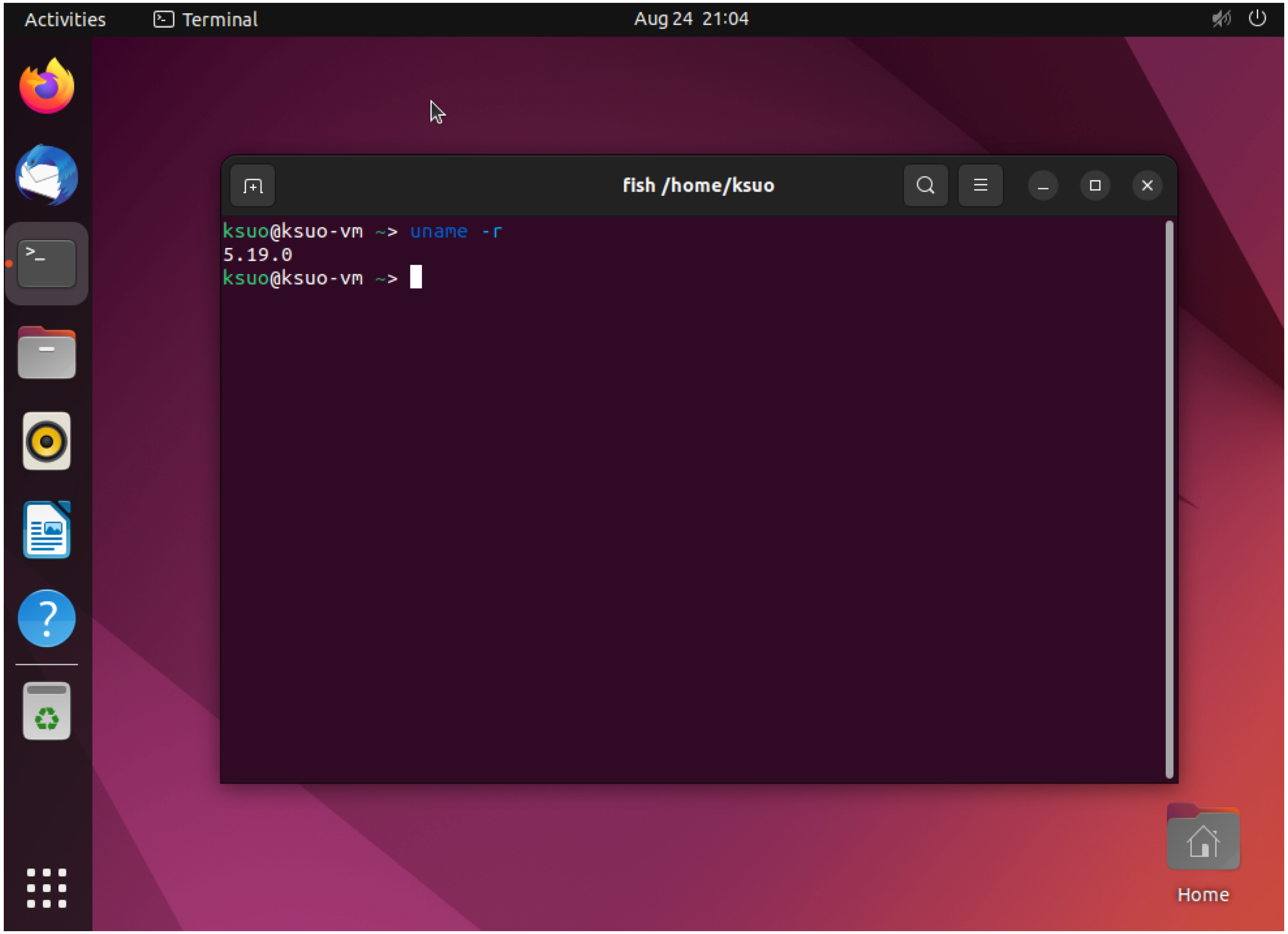


```
                  GNU GRUB  version 2.02

*Ubuntu, with Linux 5.1.0
 Ubuntu, with Linux 5.1.0 (recovery mode)
 Ubuntu, with Linux 5.0.0-25-generic
 Ubuntu, with Linux 5.0.0-25-generic (recovery mode)
 Ubuntu, with Linux 5.0.0-23-generic
 Ubuntu, with Linux 5.0.0-23-generic (recovery mode)
```

Compile the kernel

1. download the kernel source code
2. unzip the file
3. use *$ make localmodconfig* 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

Screenshot of my new kernel



```
ksuo@ksuo-vm ~> uname -r
5.19.0
ksuo@ksuo-vm ~>
```

Outline

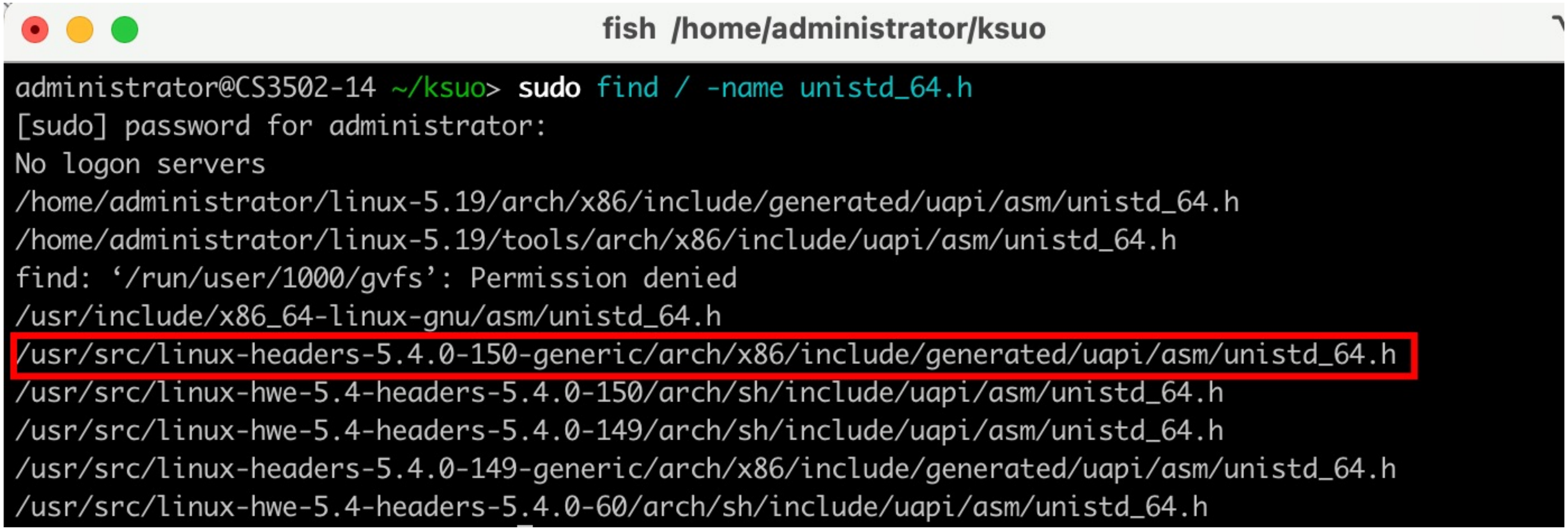
- Part A: create a VM and compile your kernel
- **Part B: add a new system call into the Linux kernel**

Define your system call

- Step 1: Check the available system call number
- Step 2: Create a kernel module syscall
- Step 3: Define the Makefile
- Step 4: Compile and enable the module syscall
- Step 5: write a test program to test your system call

Step 1: Check the available system call number

$ sudo find / -name unistd_64.h



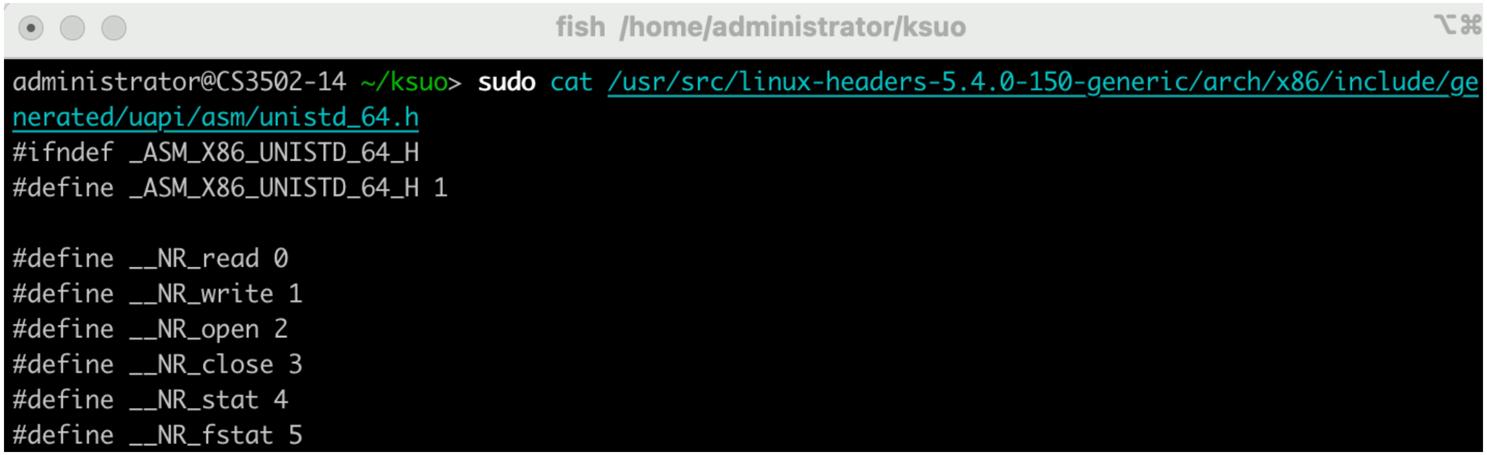
fish /home/administrator/ksuo

```
administrator@CS3502-14 ~/ksuo> sudo find / -name unistd_64.h
[sudo] password for administrator:
No logon servers
/home/administrator/linux-5.19/arch/x86/include/generated/uapi/asm/unistd_64.h
/home/administrator/linux-5.19/tools/arch/x86/include/uapi/asm/unistd_64.h
find: ‘/run/user/1000/gvfs’: Permission denied
/usr/include/x86_64-linux-gnu/asm/unistd_64.h
/usr/src/linux-headers-5.4.0-150-generic/arch/x86/include/generated/uapi/asm/unistd_64.h
/usr/src/linux-hwe-5.4-headers-5.4.0-150/arch/sh/include/uapi/asm/unistd_64.h
/usr/src/linux-hwe-5.4-headers-5.4.0-149/arch/sh/include/uapi/asm/unistd_64.h
/usr/src/linux-headers-5.4.0-149-generic/arch/x86/include/generated/uapi/asm/unistd_64.h
/usr/src/linux-hwe-5.4-headers-5.4.0-60/arch/sh/include/uapi/asm/unistd_64.h
```

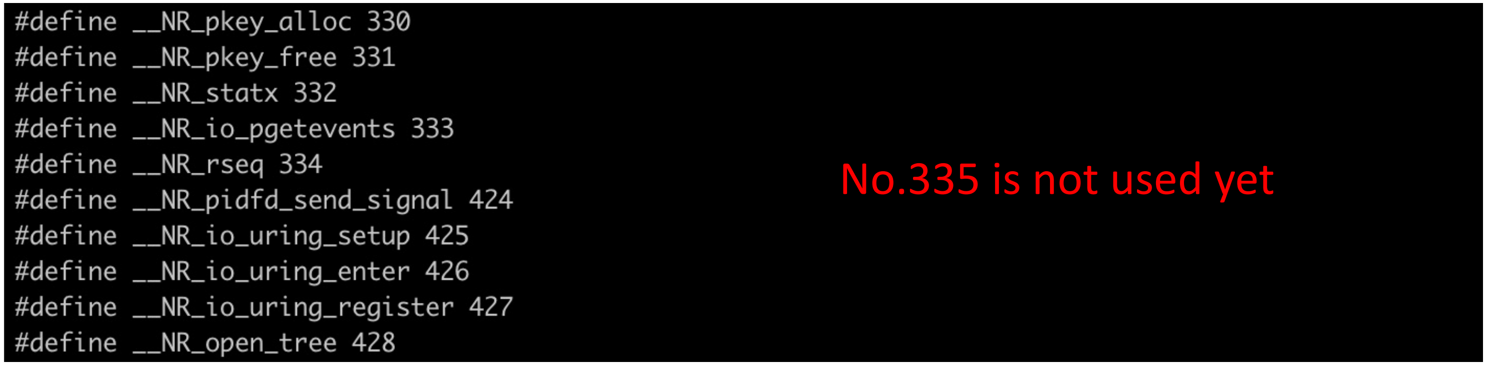
As my current Linux version is 5.4.0-150-generic, so I check the above one.

Step 1: Check the available system call number

$ sudo cat /usr/src/linux-headers-5.4.0-150-generic/arch/x86/include/generated/uapi/asm/unistd_64.h



```
administrator@CS3502-14 ~/ksuo> sudo cat /usr/src/linux-headers-5.4.0-150-generic/arch/x86/include/ge
nerated/uapi/asm/unistd_64.h
#ifndef _ASM_X86_UNISTD_64_H
#define _ASM_X86_UNISTD_64_H 1

#define __NR_read 0
#define __NR_write 1
#define __NR_open 2
#define __NR_close 3
#define __NR_stat 4
#define __NR_fstat 5
```

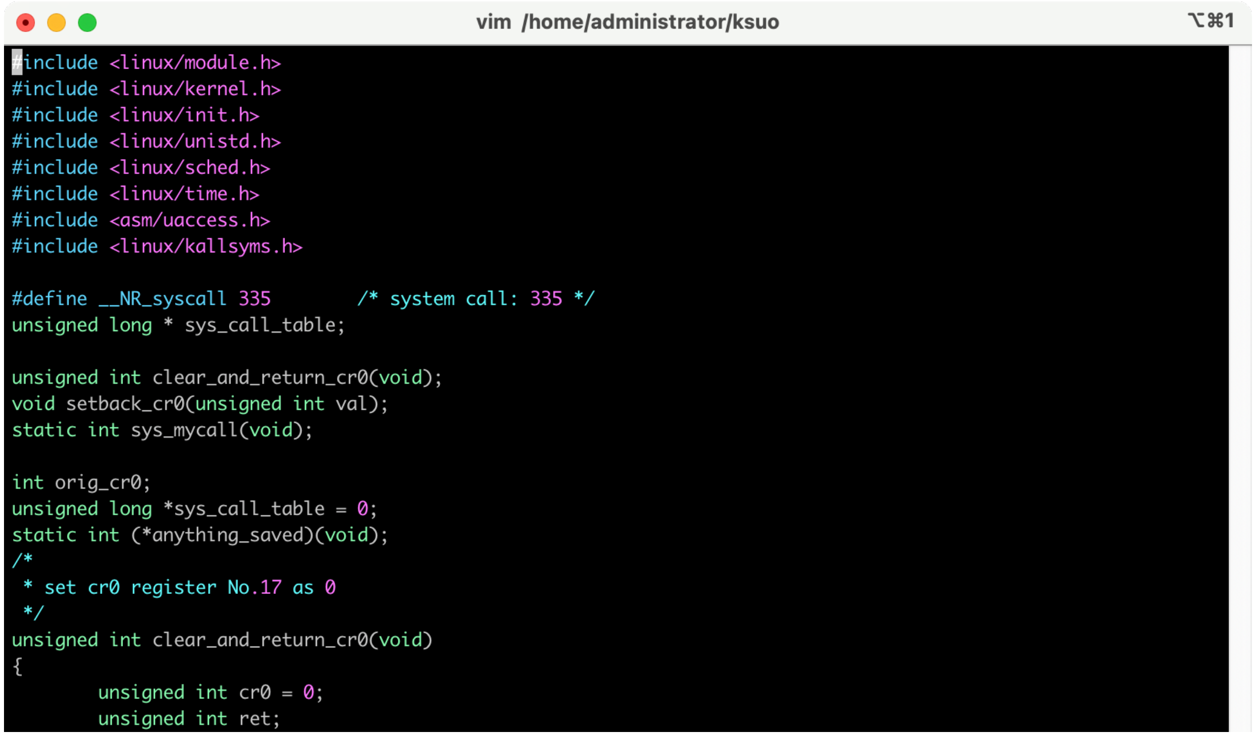


```
#define __NR_pkey_alloc 330
#define __NR_pkey_free 331
#define __NR_statx 332
#define __NR_io_pgetevents 333
#define __NR_rseq 334
#define __NR_pidfd_send_signal 424
#define __NR_io_uring_setup 425
#define __NR_io_uring_enter 426
#define __NR_io_uring_register 427
#define __NR_open_tree 428
```

No.335 is not used yet

Step 2: Create a kernel module syscall

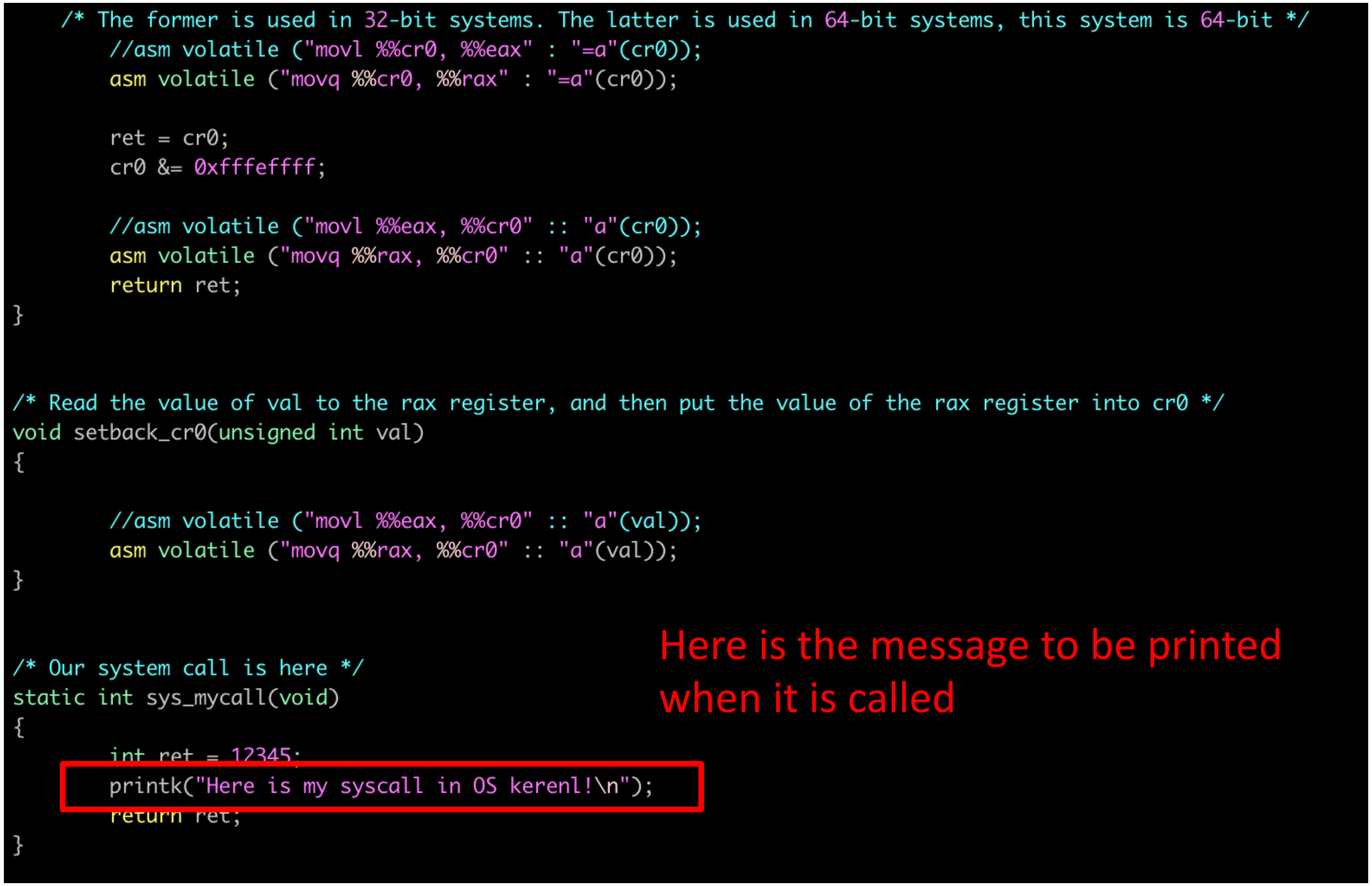
$ vim syscall.c



```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/unistd.h>
#include <linux/sched.h>
#include <linux/time.h>
#include <asm/uaccess.h>
#include <linux/kallsyms.h>

#define __NR_syscall 335        /* system call: 335 */
unsigned long * sys_call_table;

unsigned int clear_and_return_cr0(void);
void setback_cr0(unsigned int val);
static int sys_mycall(void);

int orig_cr0;
unsigned long *sys_call_table = 0;
static int (*anything_saved)(void);
/*
 * set cr0 register No.17 as 0
 */
unsigned int clear_and_return_cr0(void)
{
        unsigned int cr0 = 0;
        unsigned int ret;
```

https://github.com/kevinsuo/CS3502/blob/master/syscall.c

Step 2: Create a kernel module syscall

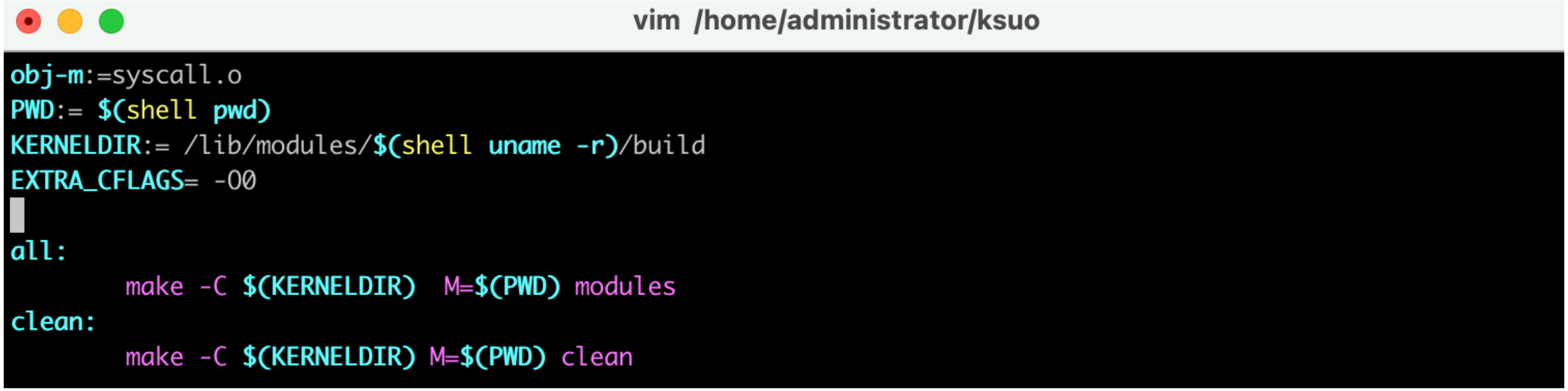


```
    /* The former is used in 32-bit systems. The latter is used in 64-bit systems, this system is 64-bit */
        //asm volatile ("movl %%cr0, %%eax" : "=a"(cr0));
        asm volatile ("movq %%cr0, %%rax" : "=a"(cr0));

        ret = cr0;
        cr0 &= 0xfffeffff;

        //asm volatile ("movl %%eax, %%cr0" :: "a"(cr0));
        asm volatile ("movq %%rax, %%cr0" :: "a"(cr0));
        return ret;
}


/* Read the value of val to the rax register, and then put the value of the rax register into cr0 */
void setback_cr0(unsigned int val)
{

        //asm volatile ("movl %%eax, %%cr0" :: "a"(val));
        asm volatile ("movq %%rax, %%cr0" :: "a"(val));
}


/* Our system call is here */
static int sys_mycall(void)
{
        int ret = 12345;
        printk("Here is my syscall in OS kerenl!\n");
        return ret;
}
```

Here is the message to be printed when it is called

https://github.com/kevinsuo/CS3502/blob/master/syscall.c

Step 3: Define the Makefile

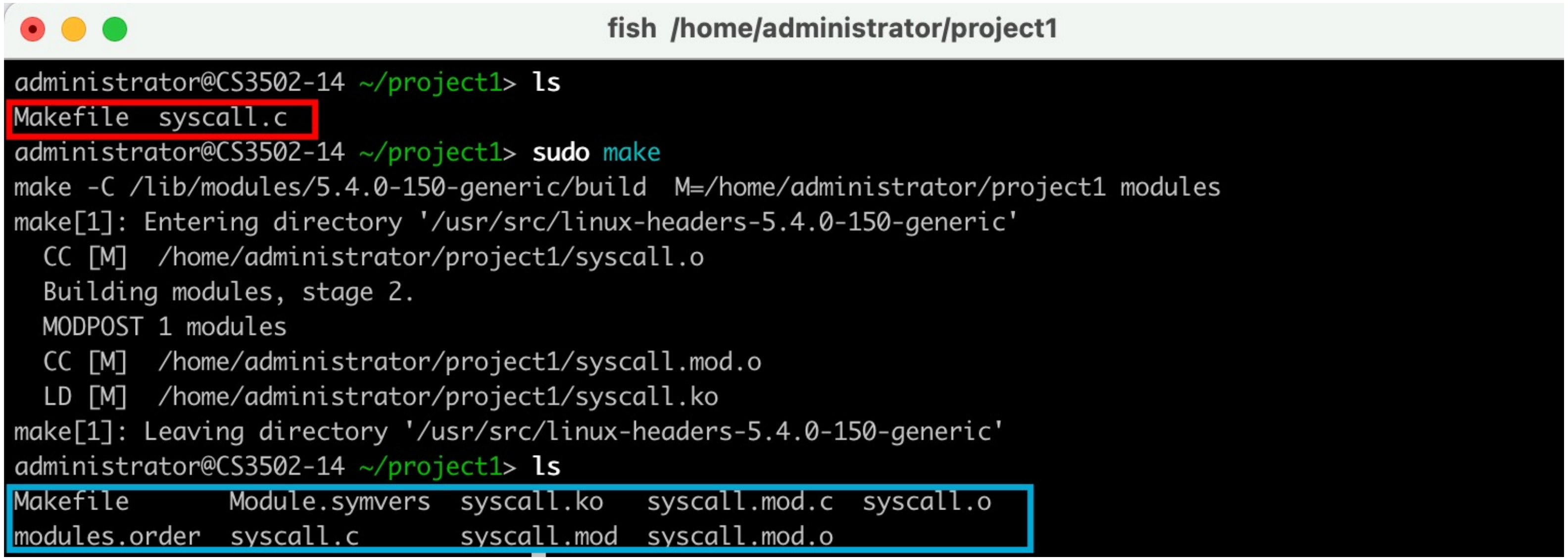


```
obj-m:=syscall.o
PWD:= $(shell pwd)
KERNELDIR:= /lib/modules/$(shell uname -r)/build
EXTRA_CFLAGS= -O0

all:
        make -C $(KERNELDIR)  M=$(PWD) modules
clean:
        make -C $(KERNELDIR) M=$(PWD) clean
```

https://github.com/kevinsuo/CS3502/blob/master/Makefile

Step 4: Compile and enable the module syscall

$ sudo make



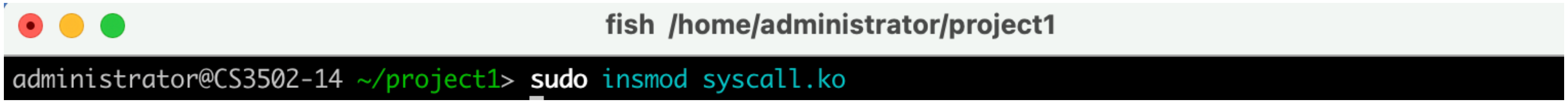
fish /home/administrator/project1

```
administrator@CS3502-14 ~/project1> ls
Makefile  syscall.c
administrator@CS3502-14 ~/project1> sudo make
make -C /lib/modules/5.4.0-150-generic/build  M=/home/administrator/project1 modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-150-generic'
  CC [M]  /home/administrator/project1/syscall.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/administrator/project1/syscall.mod.o
  LD [M]  /home/administrator/project1/syscall.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-150-generic'
administrator@CS3502-14 ~/project1> ls
Makefile       Module.symvers  syscall.ko   syscall.mod.c  syscall.o
modules.order  syscall.c       syscall.mod  syscall.mod.o
```

The red and blue parts are before and after the compiling

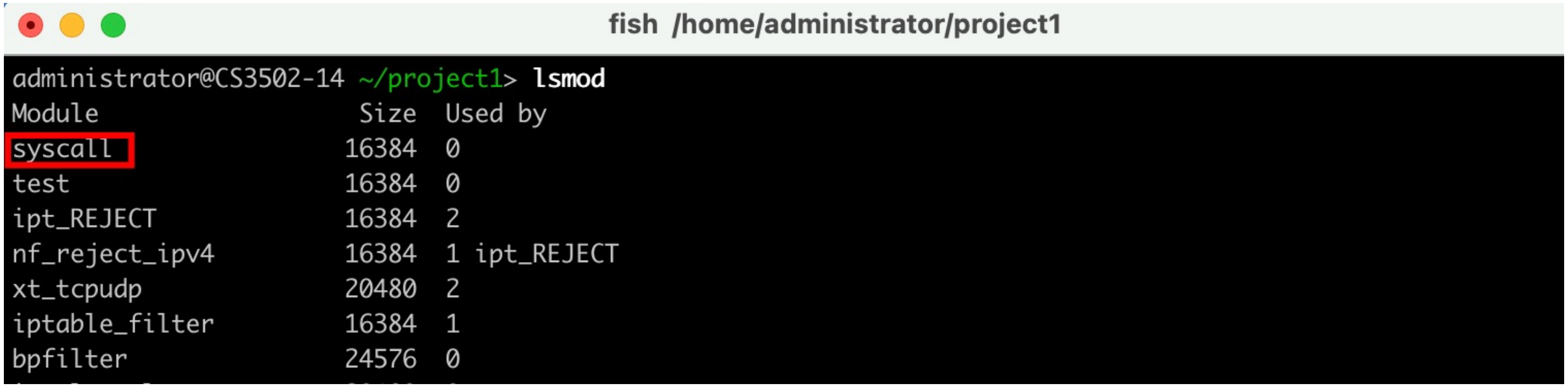
Step 4: Compile and enable the module syscall

$ sudo insmod syscall.ko



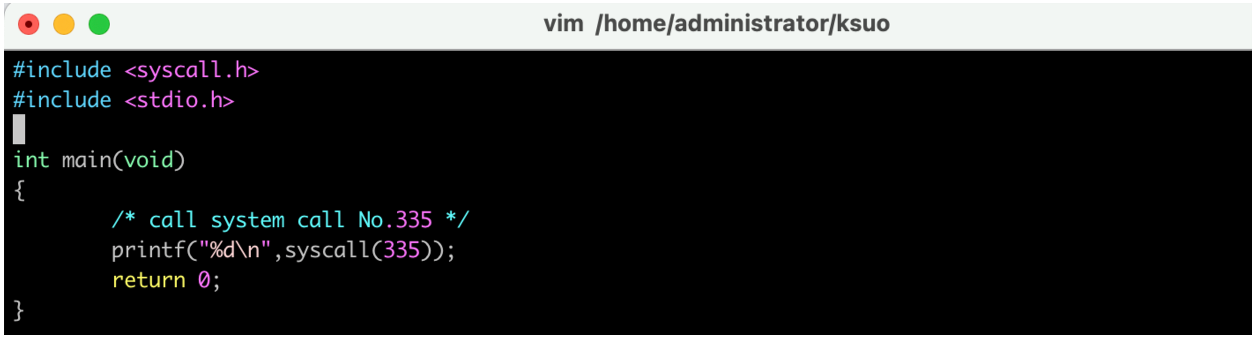
```
administrator@CS3502-14 ~/project1> sudo insmod syscall.ko
```

You can use the $ lsmod to check the enabled modules:



```
administrator@CS3502-14 ~/project1> lsmod
Module                  Size  Used by
syscall                16384  0
test                   16384  0
ipt_REJECT             16384  2
nf_reject_ipv4         16384  1 ipt_REJECT
xt_tcpudp              20480  2
iptable_filter         16384  1
bpfilter               24576  0
```

Step 5: write user level app to call it



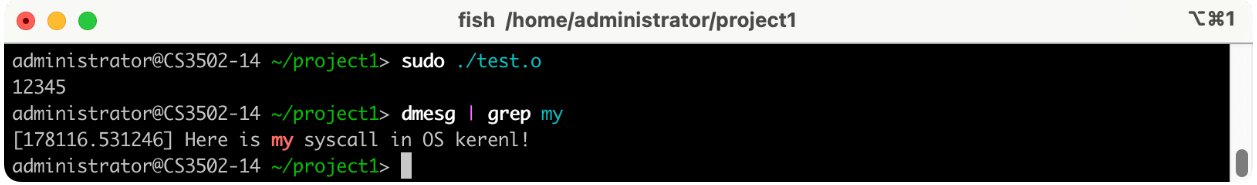
```
#include <syscall.h>
#include <stdio.h>

int main(void)
{
        /* call system call No.335 */
        printf("%d\n",syscall(335));
        return 0;
}
```

Compile and execute：

$ gcc test.c -o test.o

$ sudo ./test.o

The test program will call the new system call and output a message that you defined in step 2 at the tail of the output of dmesg (system log).

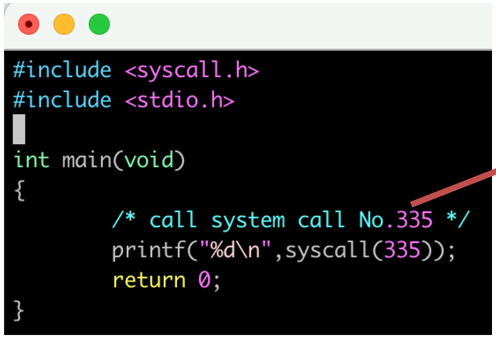


```
administrator@CS3502-14 ~/project1> sudo ./test.o
12345
administrator@CS3502-14 ~/project1> dmesg | grep my
[178116.531246] Here is my syscall in OS kerenl!
administrator@CS3502-14 ~/project1>
```

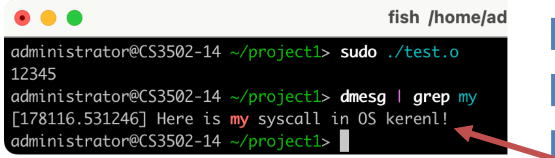
Put it all together

User space

Step 5: user call it

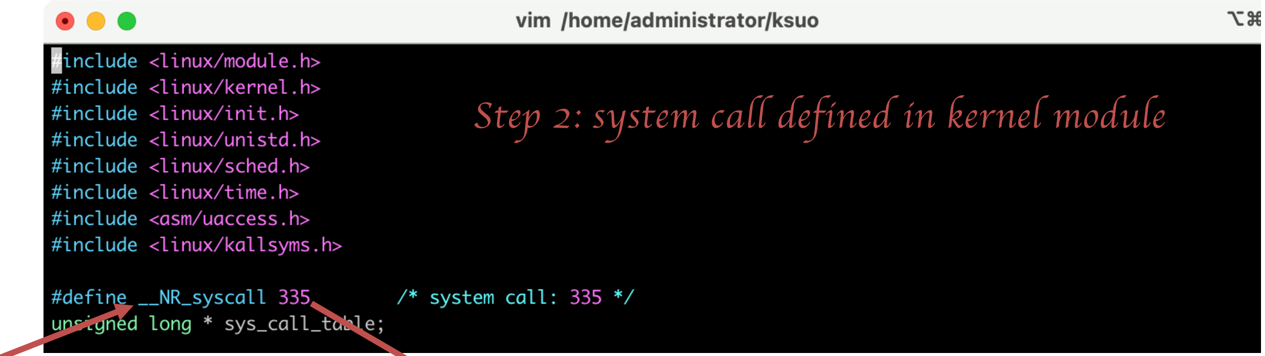


```
#include <syscall.h>
#include <stdio.h>

int main(void)
{
        /* call system call No.335 */
        printf("%d\n",syscall(335));
        return 0;
}
```



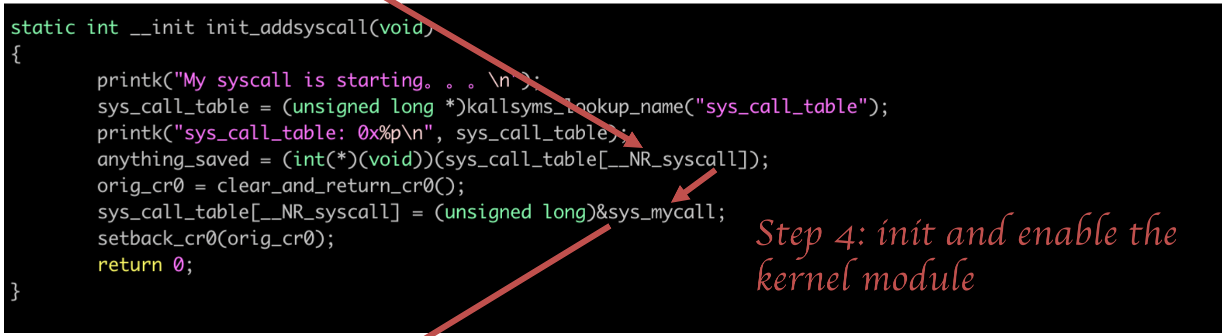
```
administrator@CS3502-14 ~/project1> sudo ./test.o
12345
administrator@CS3502-14 ~/project1> dmesg | grep my
[178116.531246] Here is my syscall in OS kerenl!
administrator@CS3502-14 ~/project1>
```

Kernel space



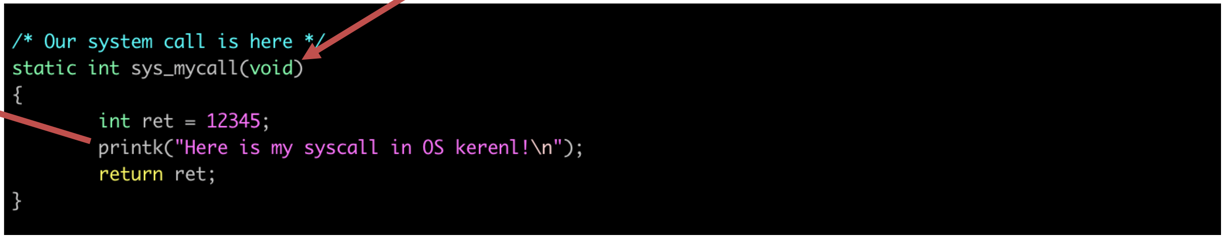
```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/unistd.h>
#include <linux/sched.h>
#include <linux/time.h>
#include <asm/uaccess.h>
#include <linux/kallsyms.h>

#define __NR_syscall 335        /* system call: 335 */
unsigned long * sys_call_table;
```

Step 2: system call defined in kernel module



```
static int __init init_addsyscall(void)
{
        printk("My syscall is starting。。。\n");
        sys_call_table = (unsigned long *)kallsyms_lookup_name("sys_call_table");
        printk("sys_call_table: 0x%p\n", sys_call_table);
        anything_saved = (int(*)(void))(sys_call_table[__NR_syscall]);
        orig_cr0 = clear_and_return_cr0();
        sys_call_table[__NR_syscall] = (unsigned long)&sys_mycall;
        setback_cr0(orig_cr0);
        return 0;
}
```

Step 4: init and enable the kernel module



```
/* Our system call is here */
static int sys_mycall(void)
{
        int ret = 12345;
        printk("Here is my syscall in OS kerenl!\n");
        return ret;
}
```