Operating System (CSC 3150)

Tutorial 2

YUANG CHEN

E-MAIL: YUANGCHEN@LINK.CUHK.EDU.CN

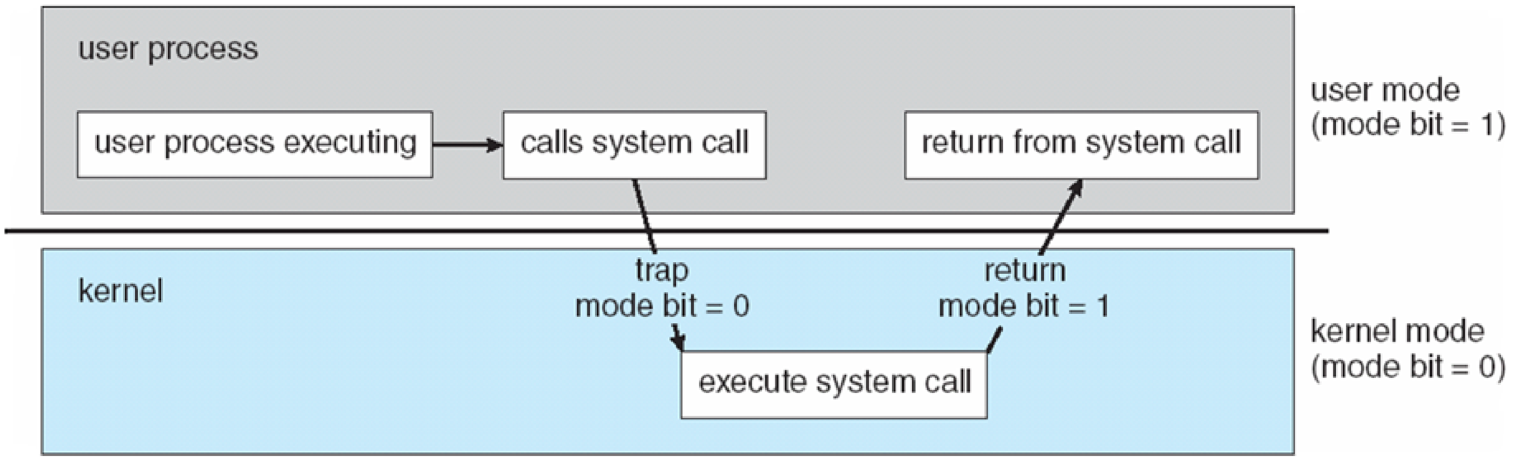
Target

In this tutorial, we will practice write system in kernel mode.

- Kernel Object
- Insert and Remove Kernel Module
- Create Kernel Thread
- Compile Kernel
- System call execution

Process

- User Mode
- Kernel Mode



Kernel Object

- A loadable kernel module (or LKM) is an object file that contains code to extend the running kernel, or so-called base kernel

- LKMs are typically used to add support for new hardware and/or file systems, or for adding system calls.

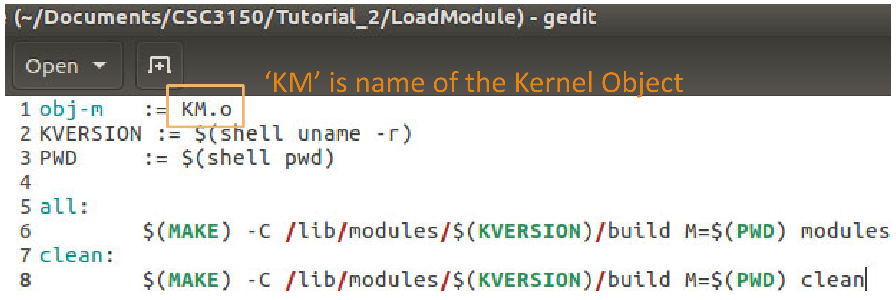
- Most current Unix-like systems support loadable kernel modules, although they might use a different name for them,
  - for example: kernel extension (kext) in MacOS

Kernel Object Compiling (Makefile)

- Makefile

http://www.cyberciti.biz/tips/compiling-linux-kernel-module.html

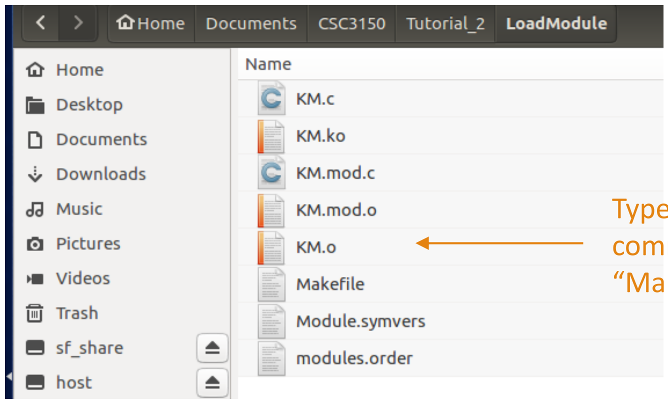
- Build kernel object



(~/Documents/CSC3150/Tutorial_2/LoadModule) - gedit

'KM' is name of the Kernel Object

```
obj-m    := KM.o
KVERSION := $(shell uname -r)
PWD      := $(shell pwd)

all:
	$(MAKE) -C /lib/modules/$(KVERSION)/build M=$(PWD) modules
clean:
	$(MAKE) -C /lib/modules/$(KVERSION)/build M=$(PWD) clean
```



```
KM.c
KM.ko
KM.mod.c
KM.mod.o
KM.o
Makefile
Module.symvers
modules.order
```

Type command "Make"

If you type command "Make clean", it will clear all built files and leave original c file and makefile .

Insert and Remove Kernel Module

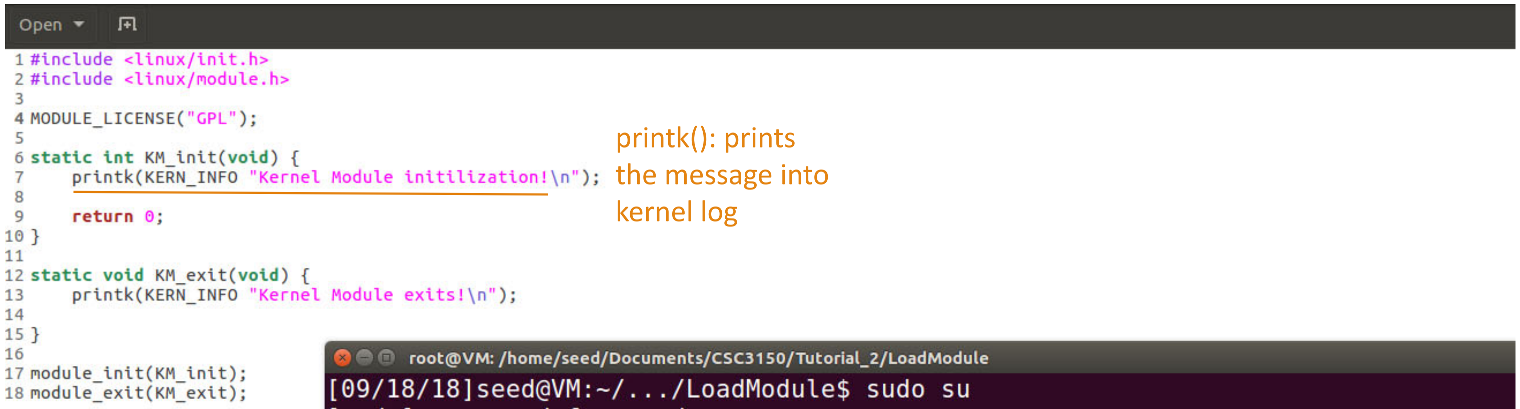
- Before insert the kernel object, you have to sign in the root account.
  - $ sudo su

- Insert module
  - $insmod MODULE_NAME.ko

- List the module you insert
  - $lsmod
  - $lsmod | grep MODULE_NAME

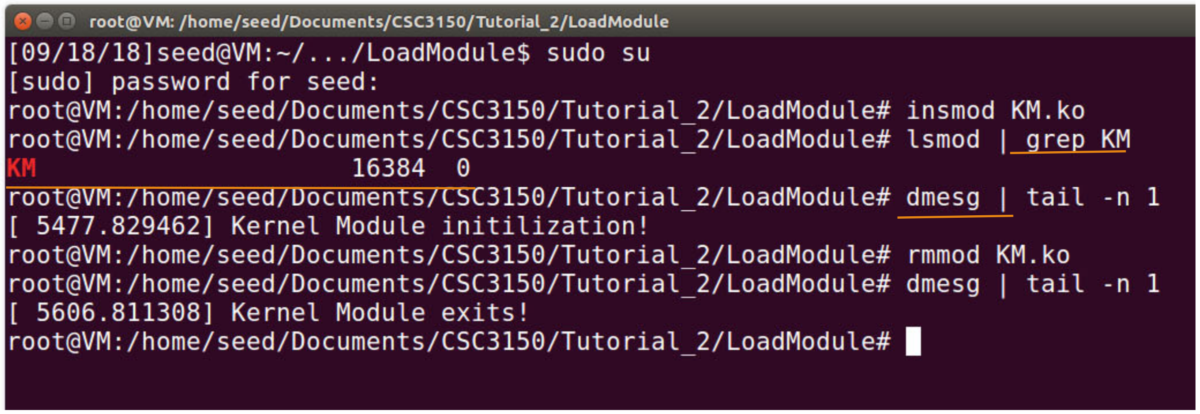
- Remember to remove your module
  - $rmmod MODULE_NAME.ko

Insert and Remove Kernel Module



```
#include <linux/init.h>
#include <linux/module.h>

MODULE_LICENSE("GPL");

static int KM_init(void) {
    printk(KERN_INFO "Kernel Module initilization!\n");

    return 0;
}

static void KM_exit(void) {
    printk(KERN_INFO "Kernel Module exits!\n");

}

module_init(KM_init);
module_exit(KM_exit);
```

printk(): prints the message into kernel log



```
[09/18/18]seed@VM:~/.../LoadModule$ sudo su
[sudo] password for seed:
root@VM:/home/seed/Documents/CSC3150/Tutorial_2/LoadModule# insmod KM.ko
root@VM:/home/seed/Documents/CSC3150/Tutorial_2/LoadModule# lsmod | grep KM
KM                     16384  0
root@VM:/home/seed/Documents/CSC3150/Tutorial_2/LoadModule# dmesg | tail -n 1
[ 5477.829462] Kernel Module initilization!
root@VM:/home/seed/Documents/CSC3150/Tutorial_2/LoadModule# rmmod KM.ko
root@VM:/home/seed/Documents/CSC3150/Tutorial_2/LoadModule# dmesg | tail -n 1
[ 5606.811308] Kernel Module exits!
root@VM:/home/seed/Documents/CSC3150/Tutorial_2/LoadModule#
```

grep: global search regular expression and print out the line

dmesg: display message buffer in kernel

- Column 1: Module Name
- Column 2: Module Size
- Column 3: Used by (denotes each module's use count and a list of referring modules)

Kernel Thread

- Kthread creation:
  - struct task_struct *kthread_create(int (*threadfn)(void *data),

    void *data,

    const char *namefmt, ...);
  - The data argument will simply be passed to the thread function.
  - The thread will not start running immediately. It will start to execute when returned task_struct is passed to wake_up_process().

- Kthread execution function:
  - int thread_function(void *data);
  - It can either call do_exit directly if it is a standalone thread for which no one will call kthread_stop()
  - Or return when 'kthread_should_stop' is true (which means kthread_stop has been called).

Kernel Thread

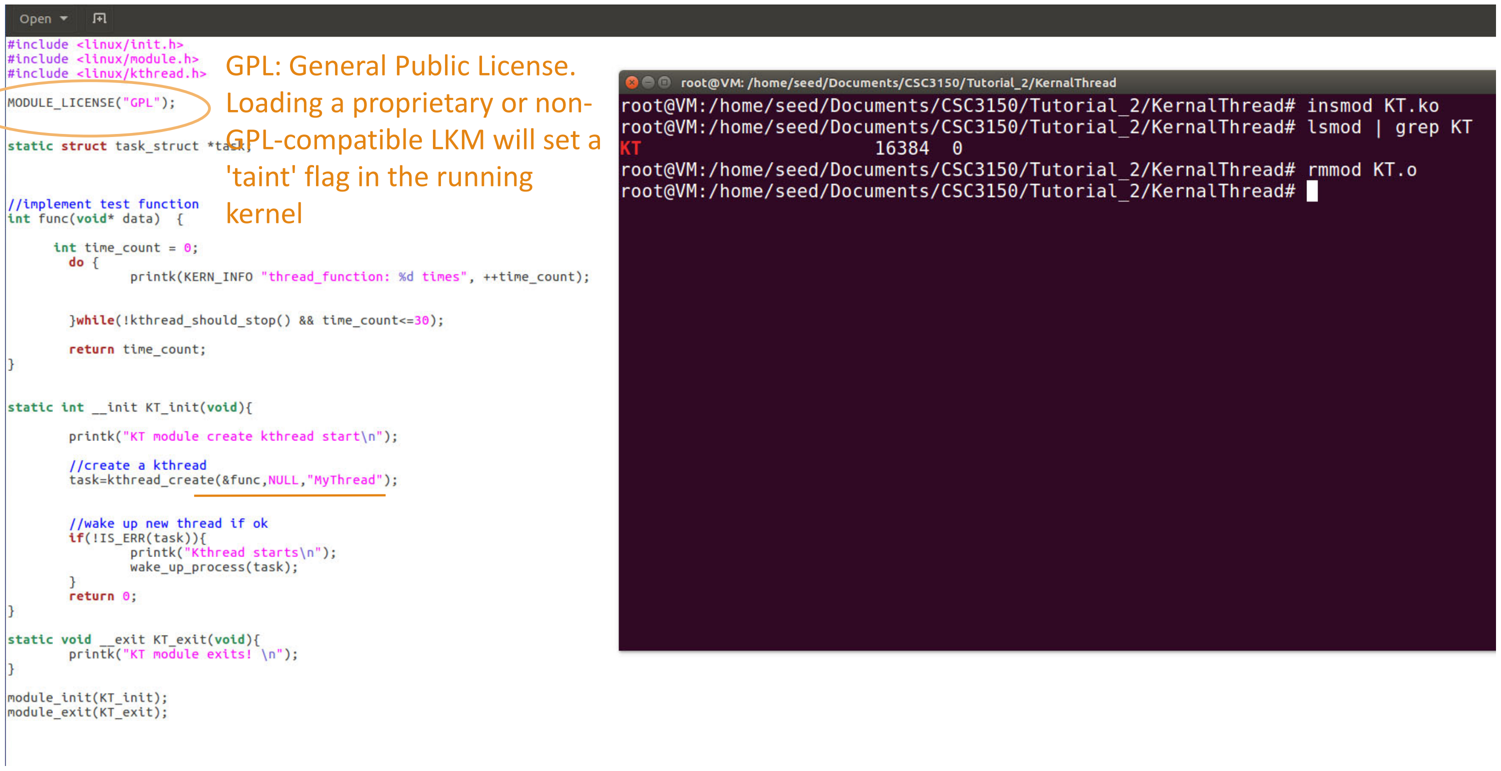
- Return value:
  - It returns task_struct when executes successfully.
  - When fails, it returns ERR_PTR

- Kthred start execution with:
  - int wake_up_process (struct task_struct * p);
  - ERR_PTR

- A convenient function which creates and starts the thread:

  -
    ```
    struct task_struct *kthread_run(        int (*threadfn)(void *data),
                                            void *data,
                                            const char *namefmt, ...);
    ```
  - Same as kthread_create() + wake_up_process()

Kernel Thread



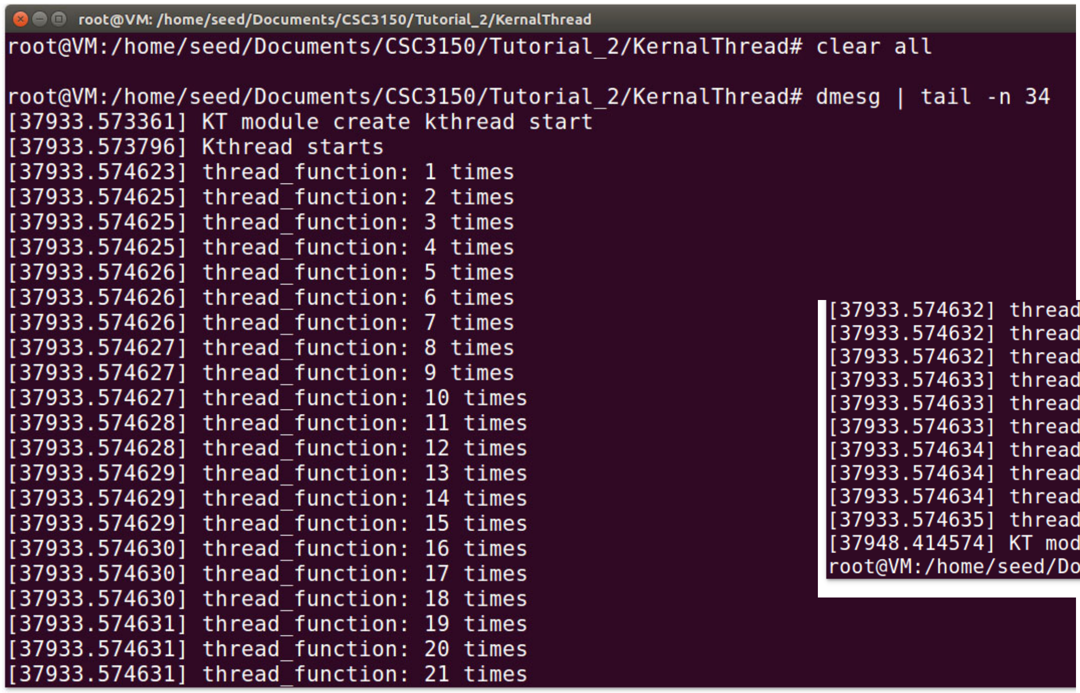
```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kthread.h>

MODULE_LICENSE("GPL");

static struct task_struct *task;


//implement test function
int func(void* data)  {

      int time_count = 0;
        do {
                printk(KERN_INFO "thread_function: %d times", ++time_count);


        }while(!kthread_should_stop() && time_count<=30);

        return time_count;
}

static int __init KT_init(void){

        printk("KT module create kthread start\n");

        //create a kthread
        task=kthread_create(&func,NULL,"MyThread");


        //wake up new thread if ok
        if(!IS_ERR(task)){
                printk("Kthread starts\n");
                wake_up_process(task);
        }
        return 0;
}

static void __exit KT_exit(void){
        printk("KT module exits! \n");
}

module_init(KT_init);
module_exit(KT_exit);
```

GPL: General Public License. Loading a proprietary or non-GPL-compatible LKM will set a 'taint' flag in the running kernel

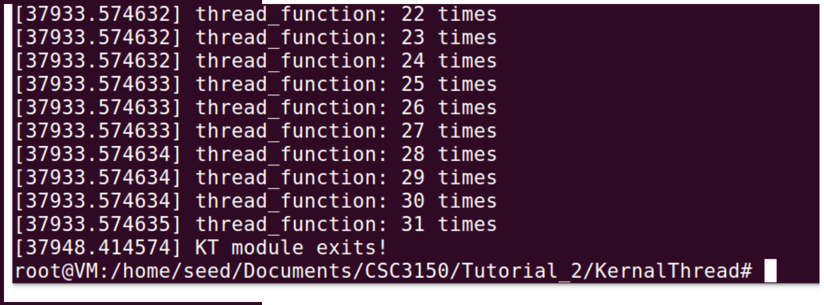
Create a kernel thread to execute func

```
root@VM:/home/seed/Documents/CSC3150/Tutorial_2/KernalThread# insmod KT.ko
root@VM:/home/seed/Documents/CSC3150/Tutorial_2/KernalThread# lsmod | grep KT
KT                     16384  0
root@VM:/home/seed/Documents/CSC3150/Tutorial_2/KernalThread# rmmod KT.o
root@VM:/home/seed/Documents/CSC3150/Tutorial_2/KernalThread# 
```

Kernel Thread



```
root@VM:/home/seed/Documents/CSC3150/Tutorial_2/KernalThread# clear all

root@VM:/home/seed/Documents/CSC3150/Tutorial_2/KernalThread# dmesg | tail -n 34
[37933.573361] KT module create kthread start
[37933.573796] Kthread starts
[37933.574623] thread_function: 1 times
[37933.574625] thread_function: 2 times
[37933.574625] thread_function: 3 times
[37933.574625] thread_function: 4 times
[37933.574626] thread_function: 5 times
[37933.574626] thread_function: 6 times
[37933.574626] thread_function: 7 times
[37933.574627] thread_function: 8 times
[37933.574627] thread_function: 9 times
[37933.574627] thread_function: 10 times
[37933.574628] thread_function: 11 times
[37933.574628] thread_function: 12 times
[37933.574629] thread_function: 13 times
[37933.574629] thread_function: 14 times
[37933.574629] thread_function: 15 times
[37933.574630] thread_function: 16 times
[37933.574630] thread_function: 17 times
[37933.574630] thread_function: 18 times
[37933.574631] thread_function: 19 times
[37933.574631] thread_function: 20 times
[37933.574631] thread_function: 21 times
```



```
[37933.574632] thread_function: 22 times
[37933.574632] thread_function: 23 times
[37933.574632] thread_function: 24 times
[37933.574633] thread_function: 25 times
[37933.574633] thread_function: 26 times
[37933.574633] thread_function: 27 times
[37933.574634] thread_function: 28 times
[37933.574634] thread_function: 29 times
[37933.574634] thread_function: 30 times
[37933.574635] thread_function: 31 times
[37948.414574] KT module exits!
root@VM:/home/seed/Documents/CSC3150/Tutorial_2/KernalThread#
```

Compile Kernel

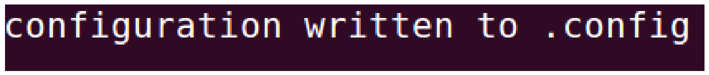
- Download source code from
  - http://www.kernel.org
  - $sudo apt-get install linux-source (Or you could type in this command to download source code directly)

- Extract the source file to /home/seed/work
  - cp KENEL_FILE.tar.xz /home/seed/work
  - cd /home/seed/work
  - $sudo tar xvf KENEL_FILE.tar.xz

- Login root account and go to kernel source directory
  - $sudo su
  - $cd /home/seed/work /KENEL_FILE

Compile Kernel

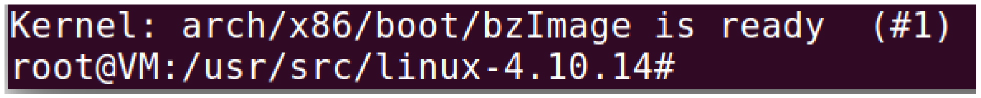
- Clean previous setting and start configuration
  - $make mrproper
  - $make clean
  - $make menuconfig
  - save the config and exit



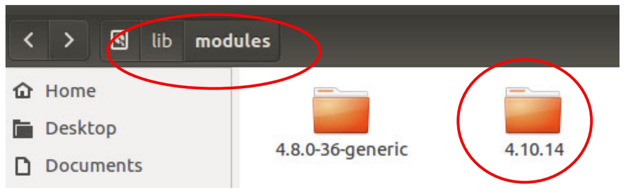
```
configuration written to .config
```

- Build kernel Image and modules
  - $make bzImage
  - $make modules
  - $make –j NUM_CORE

  (you could use this command to replace above two commands)

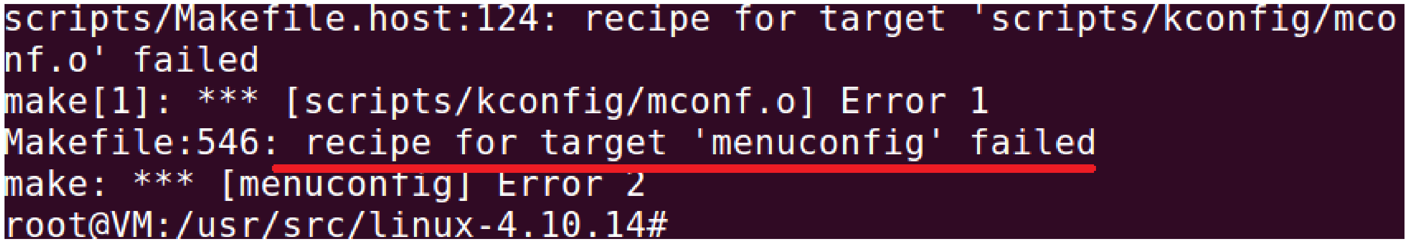


```
Kernel: arch/x86/boot/bzImage is ready  (#1)
root@VM:/usr/src/linux-4.10.14#
```



Remark: Error in menuconfig

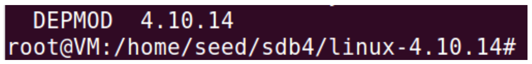
- Command "make menuconfig" does not working
  - Use command "sudo apt-get install libncurses5-dev libssl-dev" to install the tool



```
scripts/Makefile.host:124: recipe for target 'scripts/kconfig/mco
nf.o' failed
make[1]: *** [scripts/kconfig/mconf.o] Error 1
Makefile:546: recipe for target 'menuconfig' failed
make: *** [menuconfig] Error 2
root@VM:/usr/src/linux-4.10.14#
```

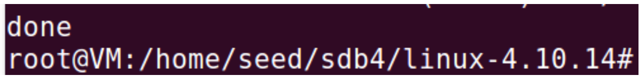
Compile Kernel

- Install kernel modules
  - $make modules_install



```
  DEPMOD  4.10.14
root@VM:/home/seed/sdb4/linux-4.10.14#
```

- Install kernel
  - $make install



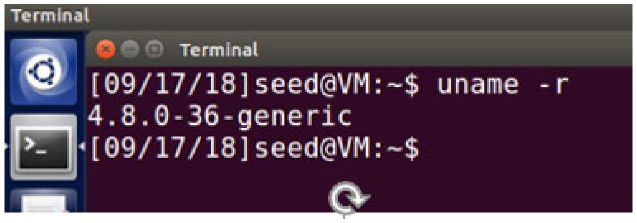
```
done
root@VM:/home/seed/sdb4/linux-4.10.14#
```

- Reboot to load new kernel
  - $reboot

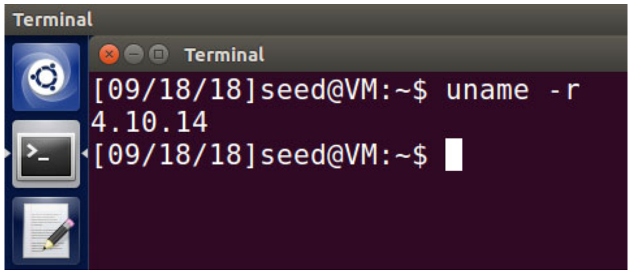
    (When rebooting, you should select the updated kernel)

Compile Kernel

- Check exiting kernel version
  - $uname -r

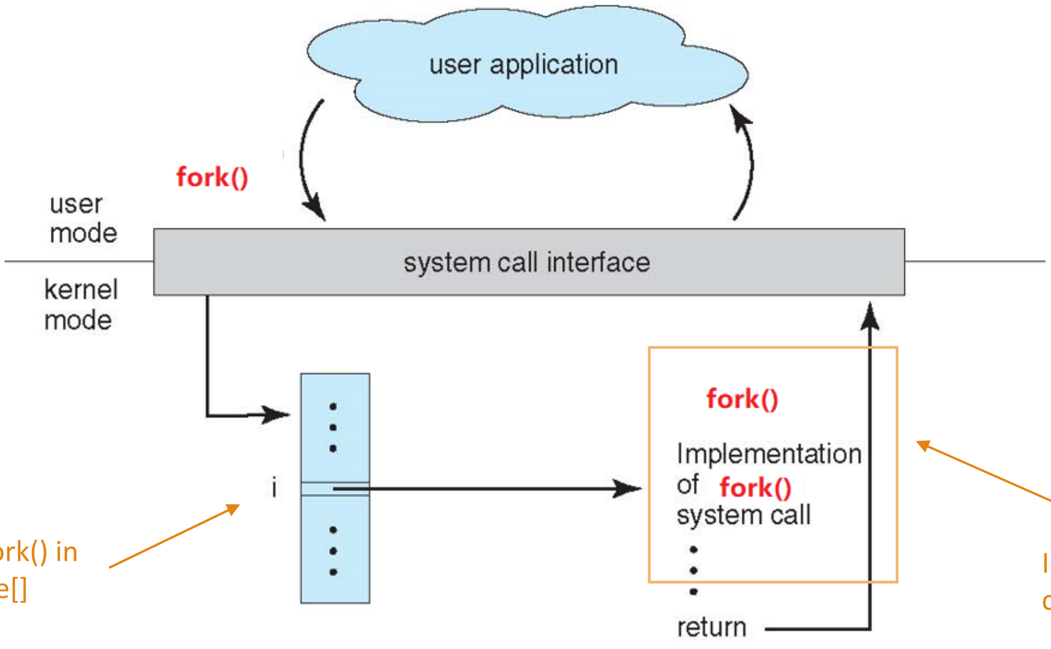


```
[09/17/18]seed@VM:~$ uname -r
4.8.0-36-generic
[09/17/18]seed@VM:~$
```



```
[09/18/18]seed@VM:~$ uname -r
4.10.14
[09/18/18]seed@VM:~$
```

System call execution (fork)



Search sys_fork() in sys_call_table[]

Implemented via calling do_fork()

System call execution (fork)

- Calls dup_task_struct(), which creates a new kernel stack, thread_info structure, and task_struct for the new process.

- Calls get_pid() to assign an available PID to the new task.

- copy_process() then either duplicates or shares open files, filesystem information, signal handlers, process address space, and namespace.

- For more details
  - https://elixir.bootlin.com/linux/v4.10.10/source/kernel/fork.c (do_fork)

Export Symbol

- EXPORT_SYMBOL() helps you provide APIs to other modules/code.
- The functions which you EXPORT are available to the other modules/code.
- Your module will not load if the it’s expecting a symbol(variable/function) and it’s not present in the kernel.

References

- Loadable module kernel
  - https://en.wikipedia.org/wiki/Loadable_kernel_module

- Kthread_create()
  - https://www.fsl.cs.sunysb.edu/kernel-api/re69.html

- Linux commands
  - http://www.runoob.com/linux/linux-command-manual.html (Chinese)

References

- Compile kernel
  - https://www.cnblogs.com/acm-icpcer/p/8029656.html (version: Linux-4.10.14, Chinese)
  - https://www.linux.com/learn/intro-to-linux/2018/4/how-compile-linux-kernel-0 (English)
  - http://www.berkes.ca/guides/linux_kernel.html (English)

- Extend storage in Virtual Box
  - https://jingyan.baidu.com/article/d45ad148a1fab869542b8073.html (Chinese)
  - http://derekmolloy.ie/resize-a-virtualbox-disk/ (English)

Thank you