

## EECS 211 Assignment 2

### EnYu Huang

**Step 1:** Follow this tutorial (or any other tutorial on kernel modules) to design a simple Kernel Module and test it:

1. Make a simple kernel module as below:

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

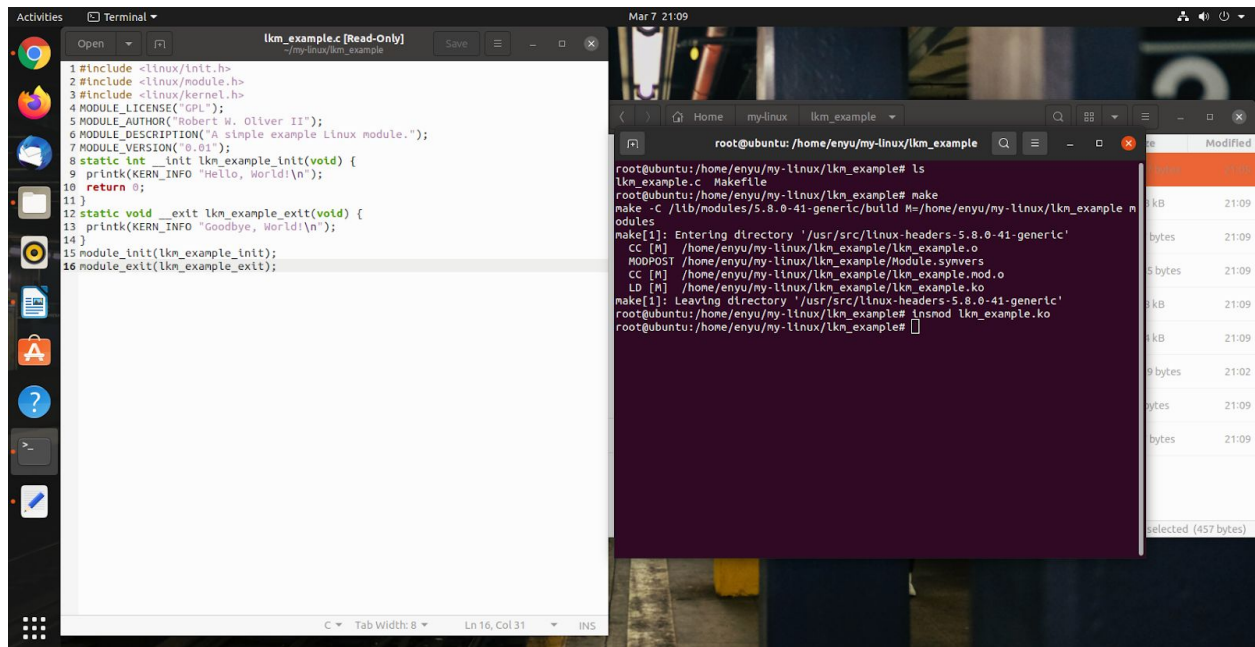
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Robert W. Oliver II");
MODULE_DESCRIPTION("A simple example Linux module.");
MODULE_VERSION("0.01");

static int __init lkm_example_init(void) {
    printk(KERN_INFO "Hello, World!\n");
    return 0;
}

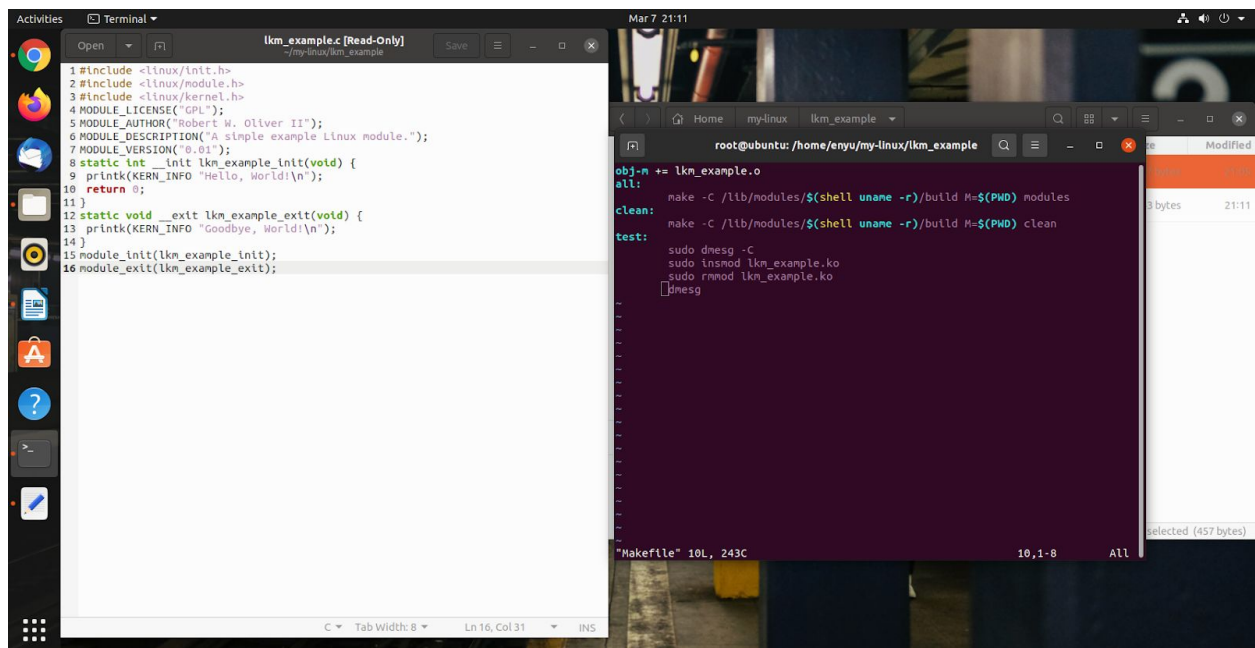
static void __exit lkm_example_exit(void) {
    printk(KERN_INFO "Goodbye, World!\n");
}

module_init(lkm_example_init);
module_exit(lkm_example_exit);
```

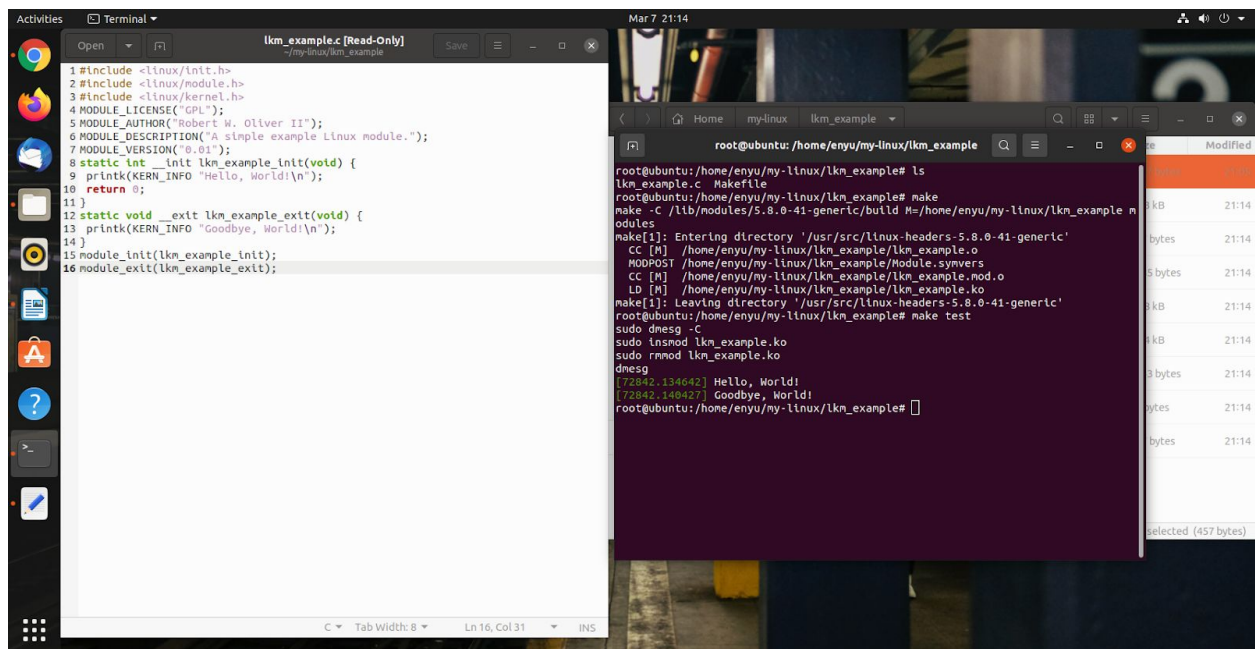
2. Write a Makefile then make it, this will generate a lkm\_example.ko, we will use it as a kernel module later on.



3. Write a test function in Makefile that includes the insmod and rmmod lkm\_example.ko so that we can do the above action with one command : \$ make test.



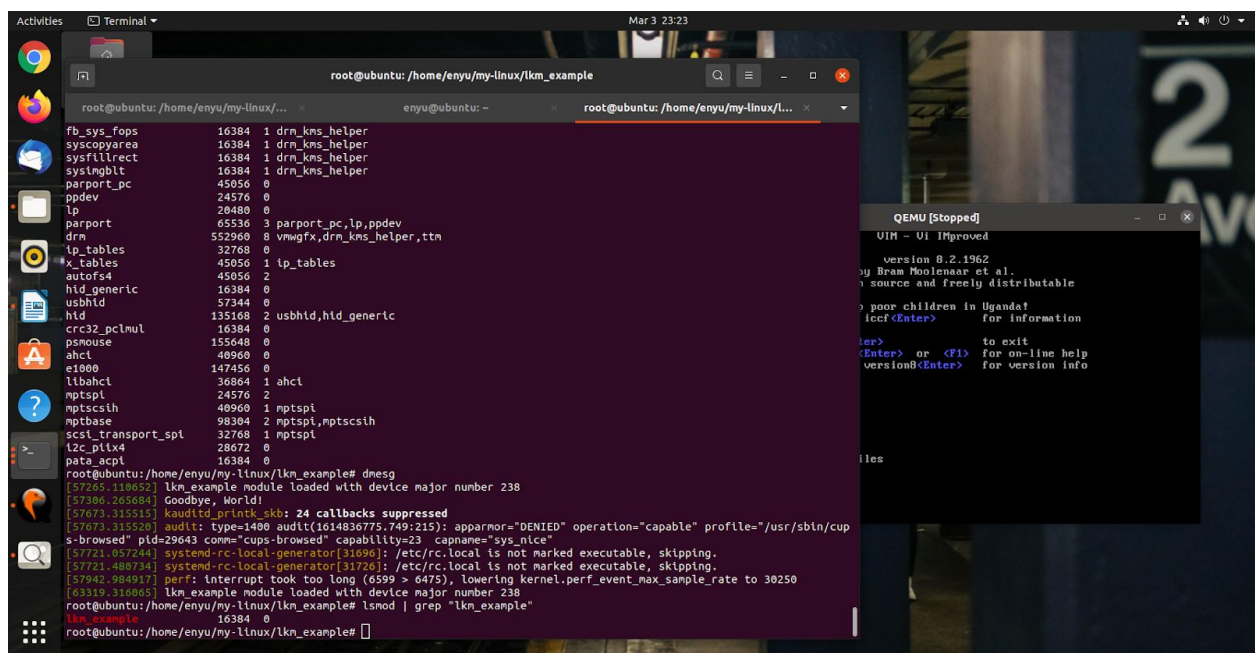
#### 4. Screenshot shows we can run this kernel module successfully



```
1 #include <linux/init.h>
2 #include <linux/module.h>
3 #include <linux/kernel.h>
4 MODULE_LICENSE("GPL");
5 MODULE_AUTHOR("Robert W. Oliver II");
6 MODULE_DESCRIPTION("A simple example Linux module.");
7 MODULE_VERSION("0.01");
8 static int __init lkm_example_init(void) {
9     printk(KERN_INFO "Hello, World!\n");
10    return 0;
11}
12 static void __exit lkm_example_exit(void) {
13    printk(KERN_INFO "Goodbye, World!\n");
14}
15 module_init(lkm_example_init);
16 module_exit(lkm_example_exit);
```

```
root@ubuntu: /home/enyu/my-linux/lkm_example
root@ubuntu: /home/enyu/my-linux/lkm_example# ls
lkm_example.c Makefile
root@ubuntu: /home/enyu/my-linux/lkm_example# make
make -C /lib/modules/5.8.0-41-generic/build M=/home/enyu/my-linux/lkm_example modules
make[1]: Entering directory '/usr/src/linux-headers-5.8.0-41-generic'
CC [M] /home/enyu/my-linux/lkm_example/lkm_example.o
MODPOST /home/enyu/my-linux/lkm_example/Module.symvers
CC [M] /home/enyu/my-linux/lkm_example/lkm_example.mod.o
LD [M] /home/enyu/my-linux/lkm_example/lkm_example.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.8.0-41-generic'
root@ubuntu: /home/enyu/my-linux/lkm_example# make test
sudo dmesg -C
sudo insmod lkm_example.ko
sudo rmmod lkm_example.ko
dmesg
[72842.134642] Hello, World!
[72842.140427] Goodbye, World!
root@ubuntu: /home/enyu/my-linux/lkm_example#
```

#### 5. In the second part, we practice the advanced method to implement kernel module



```
fb_sys_fops 16384 1 drn_kns_helper
syscopyarea 16384 1 drn_kns_helper
sysfillrect 16384 1 drn_kns_helper
sysimgblt 16384 1 drn_kns_helper
parport_pc 45056 0
ppdev 24576 0
lp 20480 0
parport 65536 3 parport_pc,lp,ppdev
drm 552960 0 vmwgfx,drn_kns_helper,ttm
lp_tables 32768 0
x_tables 45056 1 lp_tables
autofs4 45056 2
hid_generic 16384 0
usbhid 57248 0
hid 135168 2 usbhid,hid_generic
crc32_pclmul 16384 0
psmouse 155648 0
ahci 40960 0
e1000 147456 0
libahci 30864 1 ahci
mptspi 24576 2
mptscsih 40960 1 mptspi
mptbase 98304 2 mptspi,mptscsih
scsi_transport_spt 32768 1 mptspi
t2c_pllx4 28672 0
pata_acpi 16384 0
root@ubuntu: /home/enyu/my-linux/lkm_example# dmesg
[57265.110652] lkm_example module loaded with device major number 238
[57306.265684] Goodbye, World!
[57673.315515] kauditd_printk_skb: 24 callbacks suppressed
[57673.315528] audit: type=1400 audit(1614936775.740:215): apparmor="DENIED" operation="capable" profile="/usr/sbin/cups-browsed" pid=29643 comm="cups-browsed" capability=23 capname="sys_nice"
[57721.057244] systemd-rc-local-generator[31690]: /etc/rc.local is not marked executable, skipping.
[57721.480734] systemd-rc-local-generator[31720]: /etc/rc.local is not marked executable, skipping.
[57942.984917] perf: interrupt took too long (6599 > 6475), lowering kernel.perf_event_max_sample_rate to 30250
[63319.316065] lkm_example module loaded with device major number 238
root@ubuntu: /home/enyu/my-linux/lkm_example# lsmod | grep lkm_example
lkm_example 16384 0
root@ubuntu: /home/enyu/my-linux/lkm_example#
```

6. This kernel module will keep calling hello world with \$ `cat /dev/lkm_example` as show below:

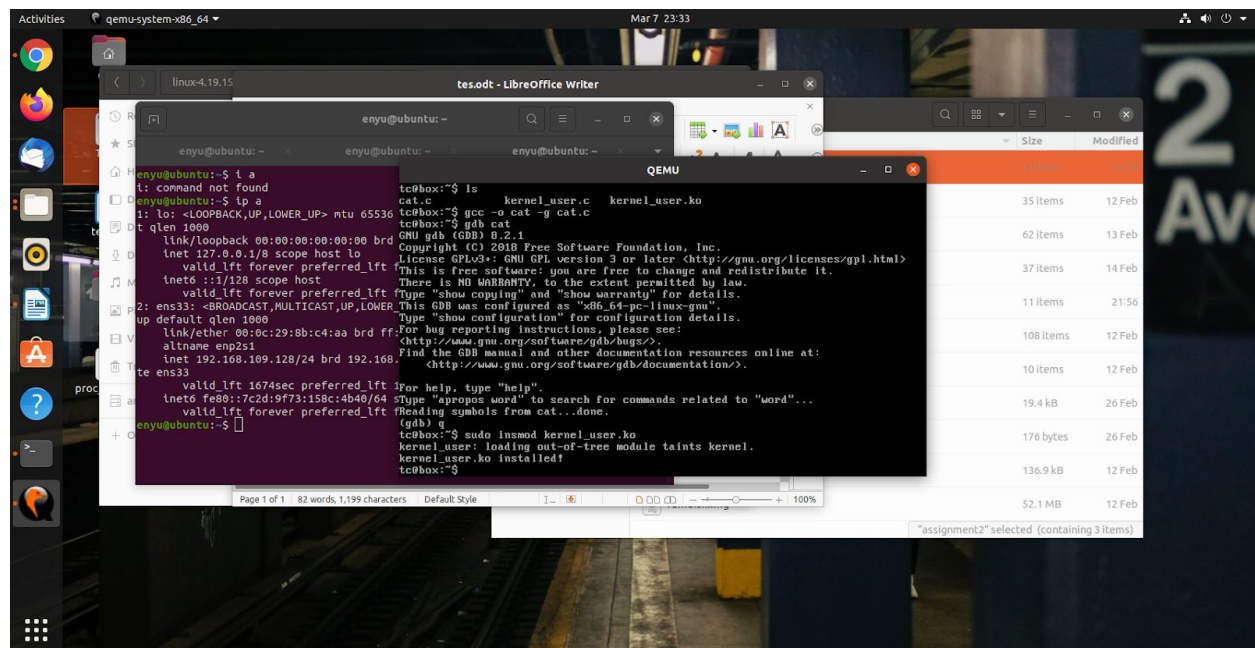
The screenshot shows a Linux desktop environment. On the left is a vertical dock with icons for various applications including a web browser, a file manager, and a terminal. The main area contains two windows. The foreground window is a terminal titled 'root@ubuntu: /home/enuy/my-linux/lkn\_example'. It displays a series of 'Hello, World!' messages, followed by a command to run 'make test'. This is followed by commands to remove the module, load it again, and check its status. The output shows the module is loaded with device major number 238. The background window is a help window titled 'MU [Stopped]' and contains text about a version 8.2.1962 release, mentioning 'Maulenaar et al.' and 'freely distributable'. It also lists some keybindings like '<F1> for on-line help'.

**Step 2(Optional):** In the previous example, you can only test the Kernel Module but you can not place breakpoints inside the Kernel Module for debugging. Use the techniques you learned in Assignment 1 to place a breakpoint in the user space (just before calling the Kernel Module APIs) and a breakpoint inside the Kernel Module that you developed. Please submit screenshots showing the CS register (if you are using x86 architecture or the correspondent register if you are using another architecture) showing that the code switched from user mode to kernel mode and back to the user mode.

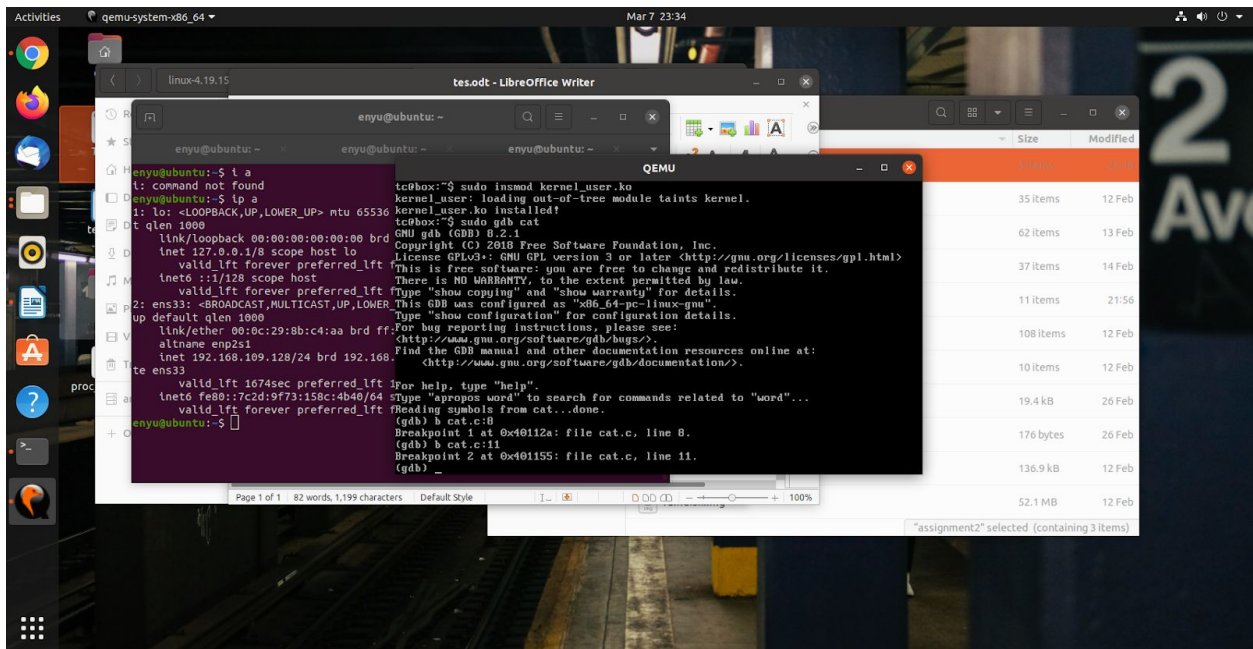


Writing a program name “cat.c” in user space to get responses from the kernel\_module, with this program, we can connect user space and kernel space. My major problem is where to set breakpoint inside cat.c and kernel\_module, after few trying, I decided to set breakpoints in cat.c at line:8 and line:11, the former is right before getting the value from kernel, the latter is at the end of the program, and a breakpoint before printk in kernel\_module, so that I can visit kernel space from user space then back to it, below is my implementation:

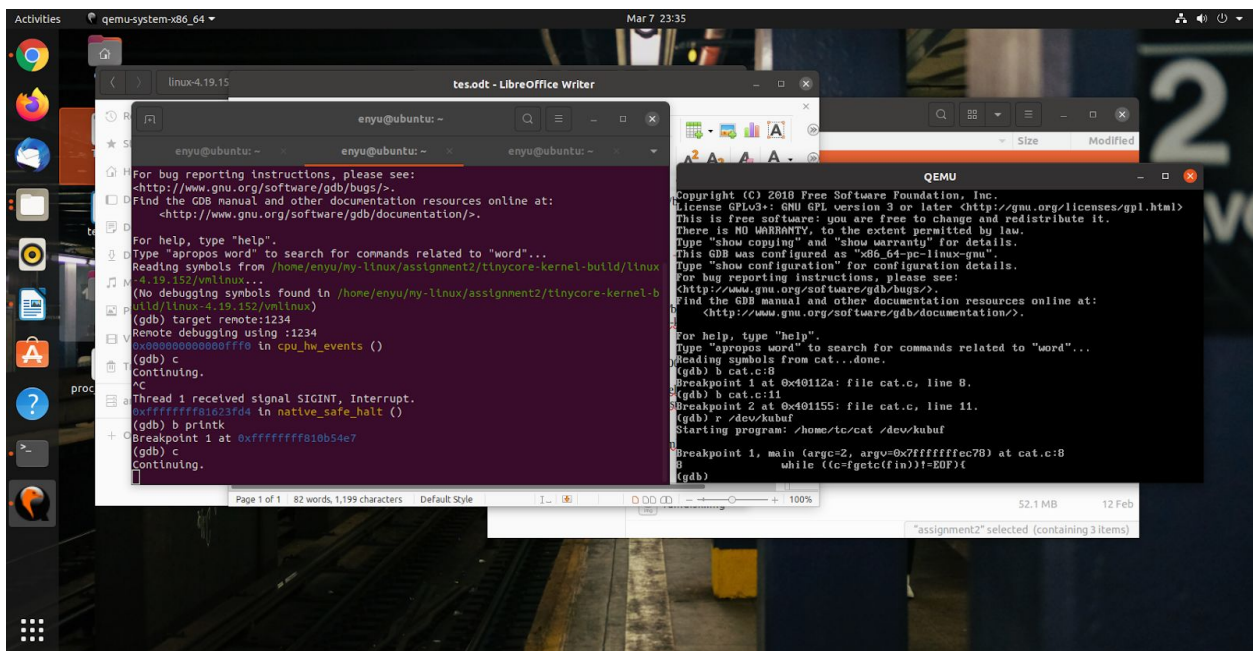
1.First open qemu and install the kernel module name “kernel\_user.ko” on it.



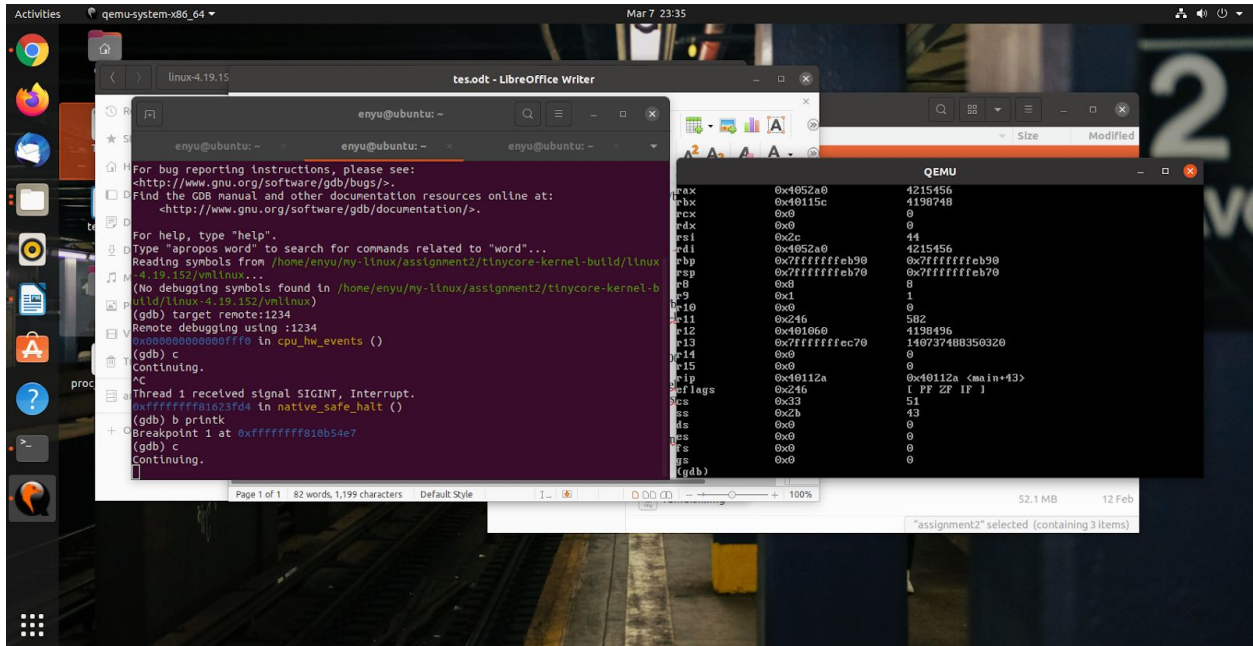
2.Set breakpoint at line:8 and line:11 in “cat.c” as I mentioned above



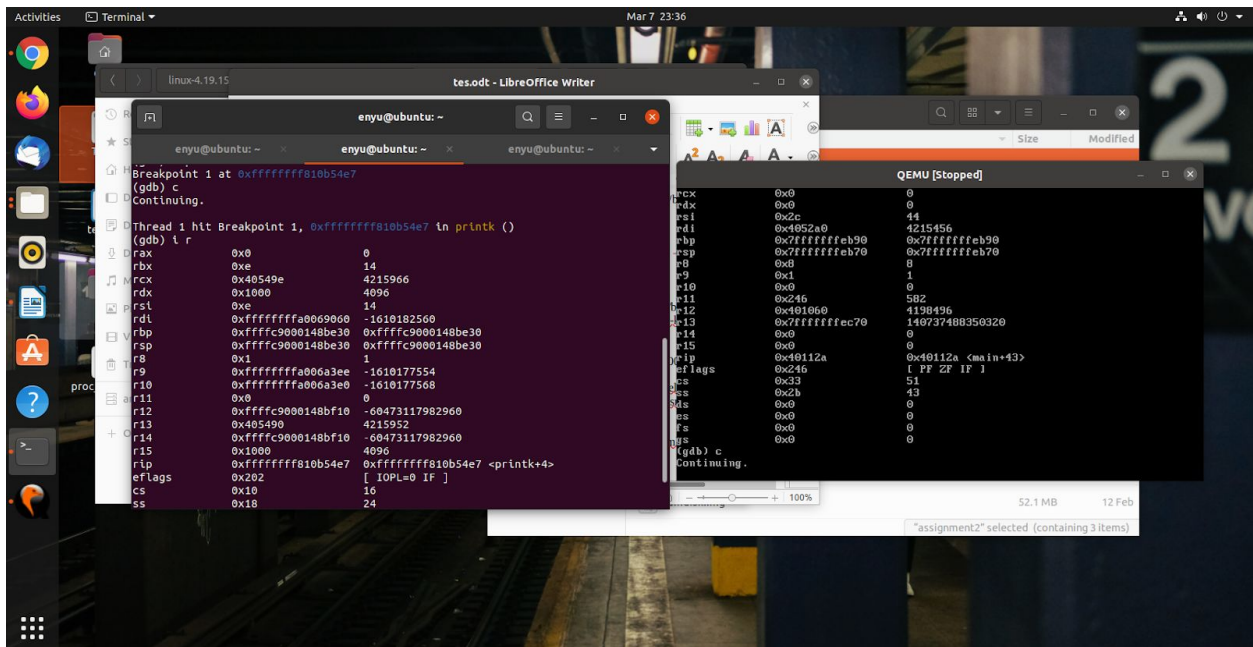
3. At kernel\_module.ko, set breakpoint before printk, so that I can stop at this point to monitor the kernel space



4. When the program stop at first breakpoint at "cat.c", by using \$info register, it shows that I am currently in the user space since cs register is 51



5. When program hit the first breakpoint in “cat.c”, pause QEMU with ctrl + c, I am now can use gdb in kernel space, use \$info register to show my current cs register value, 16, proving that it is in kernel space



6. Continue the previous breakpoint, I am now back to qemu in user space, use \$info register to show cs register with value 51 which proves that we are back to user space



