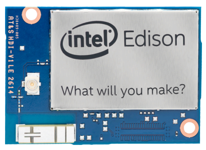


Intel® Edison Tutorial: Kernel Modules – KProbes

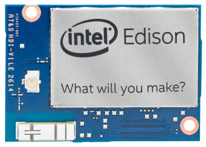
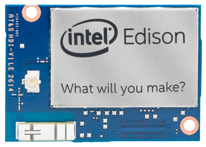


Table of Contents



Introduction

One use of kernel modules is to inspect how programs are running and what the operating system is doing. A utility called **kprobes** was developed in order to allow developers to collect debugging and performance information without disrupting to other system tasks.

This tutorial will show users how to perform the following tasks.

1. Use kprobes to monitor when the function **ping_rcv** is called.
2. Insert kprobes at different kernel routines.
3. Modify and monitor kernel routines using kprobes without recompilation of the kernel.

Prerequisite Tutorials

Users should ensure they are familiar with the documents listed below before proceeding.

1. Intel Edison Tutorial – Introduction, Linux Operating System Shell Access
2. Intel Edison Tutorial – Introduction to Linux
3. Intel Edison Tutorial – Introduction to Vim
4. Intel Edison Tutorial – Kernel Modules – Headers and Basics

Things Needed

1. 1x Intel Edison Compute Module. Operating system:
   a. Poky (Yocto Project Reference Distro) 1.7.2 edison ttyMFD2 **OR**
   b. Poky (Yocto Project Reference Distro) 1.7.3 edison ttyMFD2
2. 2 x USB 2.0 A-Male to Micro-B Cable (micro USB cable).
3. 1x powered USB hub **OR** 1x external power supply.
4. 1x Personal Computer

Kernel Modules – Inspecting Calls to ping_rcv()

This section will guide users through using kprobes to output data to the message buffer of the kernel whenever a call to a function (e.g. **ping_rcv()**) is detected.

Note: Users must ensure they have completed the tutorial labelled ***Intel Edison Tutorial – Kernel Modules – Headers and Basics*** before proceeding with this tutorial.

1. ***Backup all files present on the non-volatile flash memory of the Intel Edison to an external device such as cloud storage or personal computer***. Inserting kernel modules can cause the Linux Operating System to fail, which will prevent access to the File System. This means it is possible to lose all data onboard the Intel Edison's non-volatile flash memory if it is not backed up to an external source.

2. Open the following link on a web browser on a personal computer.

   https://drive.google.com/drive/folders/0B4NGslzPqDhvbXVkSzhPN0NUQ0k

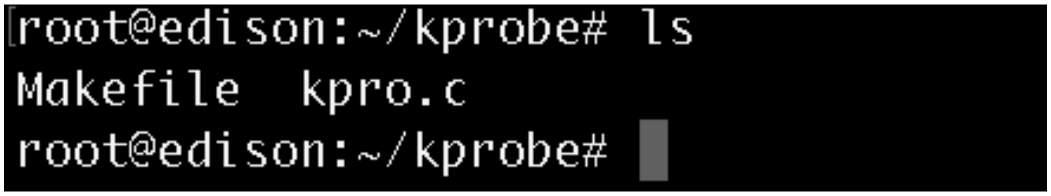
3. Download the **.zip archive** labelled **kprobe.zip**.
4. Extract the contents of **kprobe.zip** to a folder on the personal computer.
5. Access the shell program on the Intel Edison using an SSH connection. For more information, refer to the document labelled ***Intel Edison Tutorial – Introduction, Shell Access and SFTP***.

6. Create a directory labelled **kprobe** and navigate to it.

   ```
   $ mkdir ~/kprobe
   $ cd ~/kprobe
   ```

7. Transfer the files present in the folder labelled **kprobe** from the personal computer to the directory created earlier **(~/kprobe)** Intel Edison Compute Module using SFTP. For more information, refer to the document labelled ***Intel Edison Tutorial – Introduction, Shell Access and SFTP***.
8. Access the shell program on the Intel Edison using an SSH connection.
9. Change the current working directory to **~/kprobe/** and list the contents. It should match the screenshot below.

   ```
   $ cd ~/kprobe
   $ ls
   ```



```
root@edison:~/kprobe# ls
Makefile  kpro.c
root@edison:~/kprobe#
```

Figure 1: Contents of kprobe directory

10. Inspect the contents of the C-code source file **kpro.c**.

    $ cat kpro.c

11. Kprobes enable monitoring and modification of various kernel functions to observe operation and debug the kernel without needing to recompile the kernel. A kprobe can be inserted at any address in the kernel. It contains a pre-handler and a post-handler. These handler functions can perform any work required by the developer.

    Examine the functions **Pre_Handler** and **Post_Handler** in the file **kpro.c**.

12. When the instruction at the address associated with a kprobe is reached, the kprobe's pre-handler function is executed. Similarly, when the instruction at the address associated with the kprobe has completed execution, the kprobe's post-handler function is executed.

    Inspect the lines of code that assign functions to the variables **kp.pre_handler** and **kp.post_handler**.

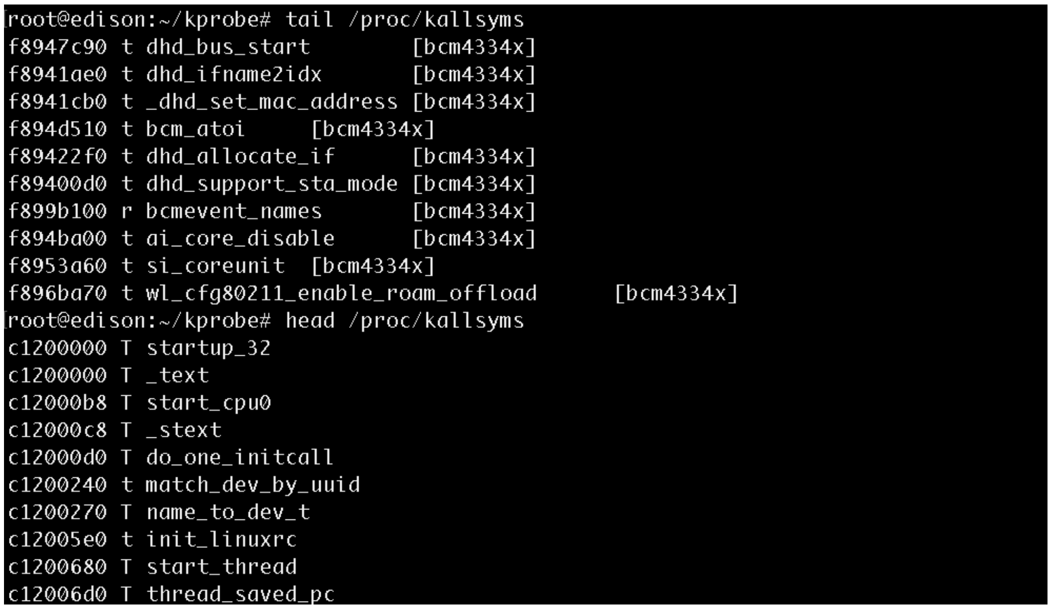
13. On removal of the module, the **unregister_kprobe()** function will remove the kprobe from the **ping_rcv()** function.

14. In order to attach a kprobe to a function, developers must find the address of the desired function to be inspected.

    The full list of functions can be found in the file **/proc/kallsyms**.

    Issue the following commands to look at the first and last ten functions available in **/proc/kallsyms**

    $ head /proc/kallsyms
    $ tail /proc/kallsyms

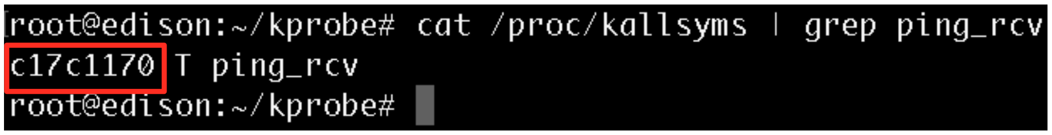


```
root@edison:~/kprobe# tail /proc/kallsyms
f8947c90 t dhd_bus_start        [bcm4334x]
f8941ae0 t dhd_ifname2idx       [bcm4334x]
f8941cb0 t _dhd_set_mac_address [bcm4334x]
f894d510 t bcm_atoi     [bcm4334x]
f89422f0 t dhd_allocate_if      [bcm4334x]
f89400d0 t dhd_support_sta_mode [bcm4334x]
f899b100 r bcmevent_names       [bcm4334x]
f894ba00 t ai_core_disable      [bcm4334x]
f8953a60 t si_coreunit  [bcm4334x]
f896ba70 t wl_cfg80211_enable_roam_offload      [bcm4334x]
root@edison:~/kprobe# head /proc/kallsyms
c1200000 T startup_32
c1200000 T _text
c12000b8 T start_cpu0
c12000c8 T _stext
c12000d0 T do_one_initcall
c1200240 t match_dev_by_uuid
c1200270 T name_to_dev_t
c12005e0 t init_linuxrc
c1200680 T start_thread
c12006d0 T thread_saved_pc
```

Figure 2: The first and last ten kernel functions present in the file /proc/kallsyms

15. Find the address of the function **ping_rcv()** by issuing the following command. Record the function address. It will be used in a later stage of this tutorial.

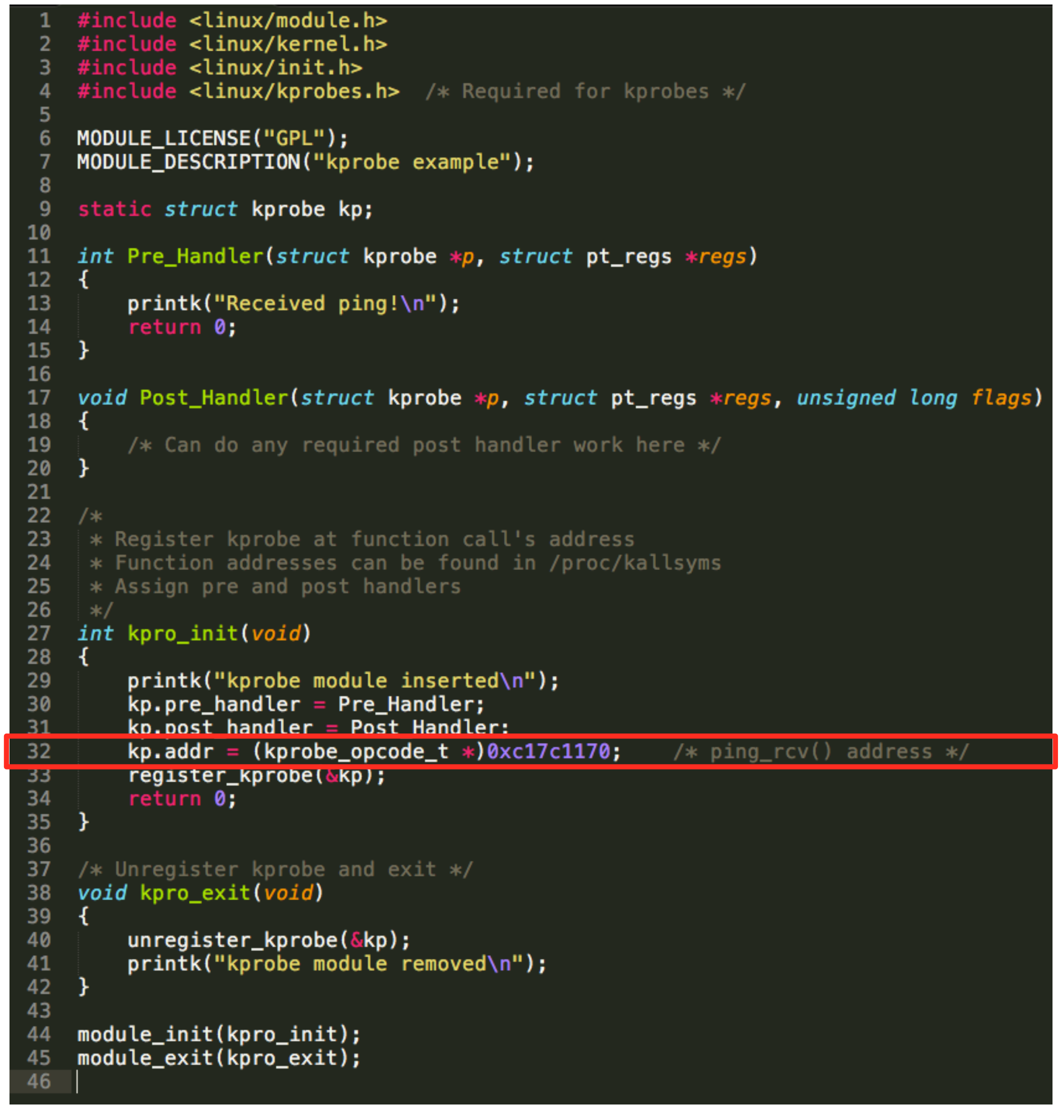
    $ cat /proc/kallsyms | grep ping_rcv



```
root@edison:~/kprobe# cat /proc/kallsyms | grep ping_rcv
c17c1170 T ping_rcv
root@edison:~/kprobe#
```

Figure 3: Address of function ping_rcv()

16. Open the **kpro.c** file and assign the address from the previous step to the variable **kp.addr**.



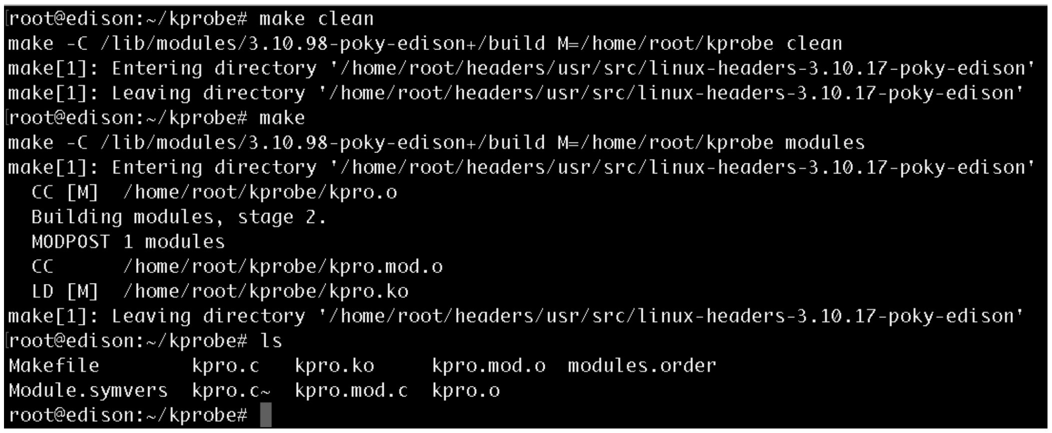
```c
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/kprobes.h>  /* Required for kprobes */

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("kprobe example");

static struct kprobe kp;

int Pre_Handler(struct kprobe *p, struct pt_regs *regs)
{
    printk("Received ping!\n");
    return 0;
}

void Post_Handler(struct kprobe *p, struct pt_regs *regs, unsigned long flags)
{
    /* Can do any required post handler work here */
}

/*
 * Register kprobe at function call's address
 * Function addresses can be found in /proc/kallsyms
 * Assign pre and post handlers
 */
int kpro_init(void)
{
    printk("kprobe module inserted\n");
    kp.pre_handler = Pre_Handler;
    kp.post_handler = Post_Handler;
    kp.addr = (kprobe_opcode_t *)0xc17c1170;    /* ping_rcv() address */
    register_kprobe(&kp);
    return 0;
}

/* Unregister kprobe and exit */
void kpro_exit(void)
{
    unregister_kprobe(&kp);
    printk("kprobe module removed\n");
}

module_init(kpro_init);
module_exit(kpro_exit);
```

Figure 4: Assigning the address of ping_rcv() to the variable kp.addr

17. Save and exit the file.
18. Issue the commands listed below.

$ make clean
$ make

On successful compilation, the make program should produce the output and files shown below.



```
root@edison:~/kprobe# make clean
make -C /lib/modules/3.10.98-poky-edison+/build M=/home/root/kprobe clean
make[1]: Entering directory '/home/root/headers/usr/src/linux-headers-3.10.17-poky-edison'
make[1]: Leaving directory '/home/root/headers/usr/src/linux-headers-3.10.17-poky-edison'
root@edison:~/kprobe# make
make -C /lib/modules/3.10.98-poky-edison+/build M=/home/root/kprobe modules
make[1]: Entering directory '/home/root/headers/usr/src/linux-headers-3.10.17-poky-edison'
  CC [M]  /home/root/kprobe/kpro.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/root/kprobe/kpro.mod.o
  LD [M]  /home/root/kprobe/kpro.ko
make[1]: Leaving directory '/home/root/headers/usr/src/linux-headers-3.10.17-poky-edison'
root@edison:~/kprobe# ls
Makefile        kpro.c   kpro.ko     kpro.mod.o  modules.order
Module.symvers  kpro.c~  kpro.mod.c  kpro.o
root@edison:~/kprobe#
```

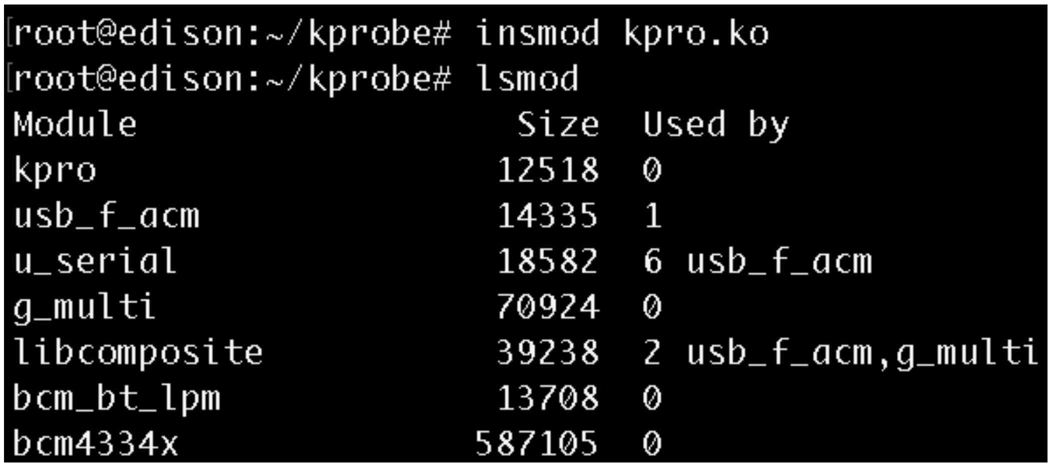
Figure 5: Files and output produced from the program 'make'

19. Issue the command below to inspect the list of modules currently added to the kernel.

    $ lsmod

20. Insert the kernel module **kpro** by issuing the **insmod** command. Check the list of kernel modules.

    $ insmod kpro.ko
    $ lsmod

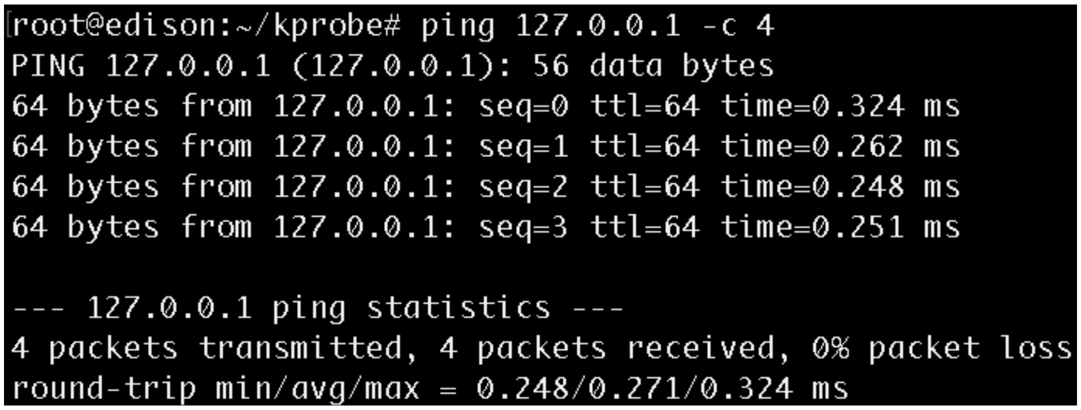


```
root@edison:~/kprobe# insmod kpro.ko
root@edison:~/kprobe# lsmod
Module                  Size  Used by
kpro                   12518  0
usb_f_acm              14335  1
u_serial               18582  6 usb_f_acm
g_multi                70924  0
libcomposite           39238  2 usb_f_acm,g_multi
bcm_bt_lpm             13708  0
bcm4334x              587105  0
```

Figure 6: Output from lsmod showing that kpro is currently loaded

21. Issue the following **ping** command.

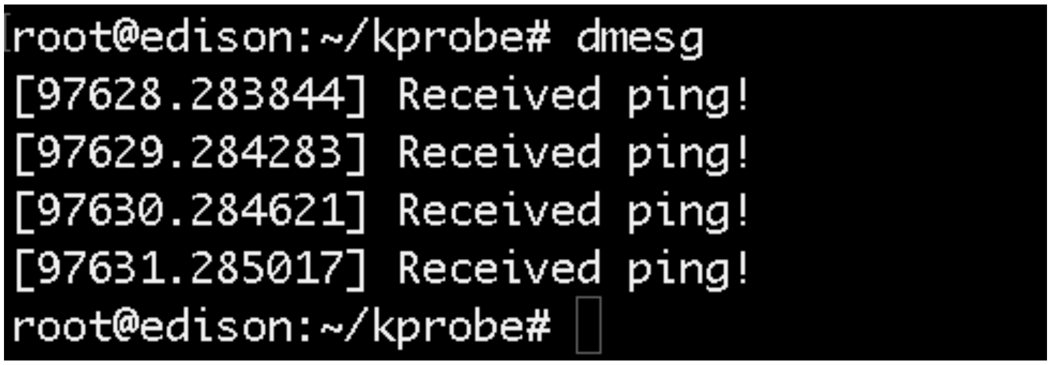
    $ ping 127.0.0.1 –c 4



```
root@edison:~/kprobe# ping 127.0.0.1 -c 4
PING 127.0.0.1 (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: seq=0 ttl=64 time=0.324 ms
64 bytes from 127.0.0.1: seq=1 ttl=64 time=0.262 ms
64 bytes from 127.0.0.1: seq=2 ttl=64 time=0.248 ms
64 bytes from 127.0.0.1: seq=3 ttl=64 time=0.251 ms

--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.248/0.271/0.324 ms
```

22. Inspect the message buffer of the kernel.

$ dmesg

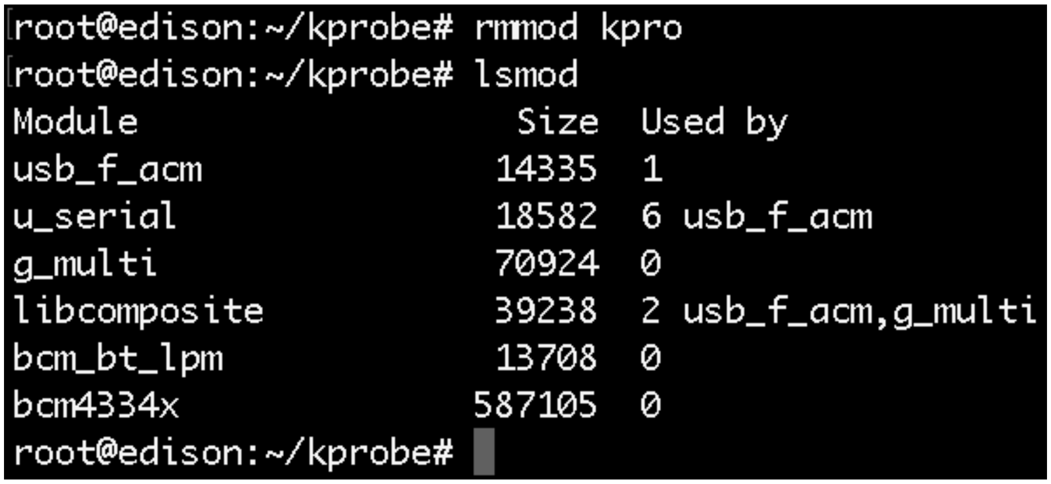


```
root@edison:~/kprobe# dmesg
[97628.283844] Received ping!
[97629.284283] Received ping!
[97630.284621] Received ping!
[97631.285017] Received ping!
root@edison:~/kprobe#
```

Figure 7: Output from dmesg showing that the kernel module functioned as expected

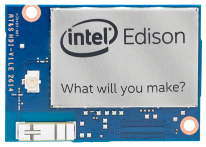
23. Remove the kernel module by issuing the following command.

$ rmmod kpro
$ lsmod



```
root@edison:~/kprobe# rmmod kpro
root@edison:~/kprobe# lsmod
Module                  Size  Used by
usb_f_acm              14335  1
u_serial               18582  6 usb_f_acm
g_multi                70924  0
libcomposite           39238  2 usb_f_acm,g_multi
bcm_bt_lpm             13708  0
bcm4334x              587105  0
root@edison:~/kprobe#
```

Figure 8: Output from lsmod after krpo has been stopped



Tasks

Task 1

Another way to detect incoming packets is to use a kprobe at the **ip_rcv()** function. This is a more general function that is called whenever an IP packet is received. It differs from **ping_rcv()** as **ping_rcv()** is only called when a **ping packet** is received.

Modify the module to register the kprobe at the **ip_rcv()** function's address.

Hint: Think about the command below. How would it need to be modified to find the address for **ip_rcv()**?

```
$ cat /proc/kallsyms | grep ping_rcv
```

Modify the **Pre_Handler** function such that it prints **"Received packet!\n"** instead of **"Received ping!\n"**

Build, insert, and remove the module. Take a screenshot of the message buffer of the kernel. Submit this screenshot.

Task 2

Modify the code such that it now:

1. Prints **"Received ping!\n"** each time **ping_rcv** is called
2. Prints **"Received packet!\n"** each time **ip_rcv** is called

This will require your code to register and unregister two kprobes.

Build, insert, and remove the module. Take a screenshot of the implementation of the function **kpro_init()** and **kpro_exit()**. Submit this screenshot.