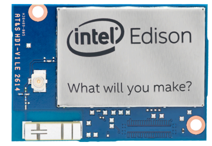
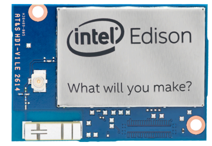


Intel® Edison Tutorial:
Kernel Modules – Headers and Basics

Table of Contents



Introduction

A loadable kernel module in Linux enables the addition of new executable code resources to the operating system kernel *at runtime*. Kernel modules provide system adaptability and capability that is essential in embedded systems. In this section, you will build, execute and observe the operation of module.

This tutorial will show users how to write, compile, insert and remove a simple kernel module. After inserting the module, users will be guided through the process of inspecting the message buffer 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

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 – Header Files

Linux kernel module source code requires access to the complete set of kernel resources available in the kernel headers. The headers in the Linux kernel define interfaces between:

A. Components of the kernel
B. The kernel and the user space

This section will provide instructions to enable access to the kernel header files. This will enable compilation of kernel modules on the Intel Edison.

Note: For this tutorial, the Intel Edison Compute Module must be running one of the two operating system versions listed below.

A. Poky (Yocto Project Reference Distro) 1.7.2 edison ttyMFD2 **OR**
B. Poky (Yocto Project Reference Distro) 1.7.3 edison ttyMFD2

This version information is available at the login screen when establishing a serial connection between a personal computer and the Intel Edison Compute Module.

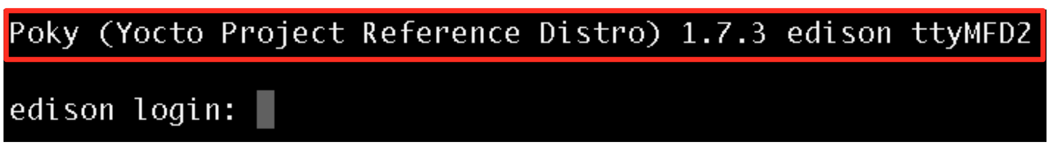


Figure 1: Gathering operating system version information from the Intel Edison Compute Module

If the Intel Edison device is not running one of these two operating systems, refer to the document labelled ***Intel Edison Tutorial – Troubleshooting Guide*** to install the latest version of the Yocto Embedded Linux Operating System on the Intel Edison Compute Module.

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 **headers.zip**.
4. Extract the contents of **headers.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 **headers** and navigate to it.

   $ mkdir ~/headers
   $ cd ~/headers

   NOTE: Ensure this directory is never deleted. If it gets deleted, kernel module development will not be possible.

7. Transfer the files present in the folder labelled **headers** from the personal computer to the directory created earlier **(~/headers)** 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 **~/headers/** and list the contents. It should match the screenshot below.

   $ cd ~/headers
   $ ls

```
root@edison:~# cd ~/headers
root@edison:~/headers# ls
clean
installheaders.sh
linux-headers-3.10.17-poky-edison_3.10.17-poky-edison-1_i386.deb
test.sh
root@edison:~/headers#
```

Figure 2: Required headers for kernel module development

10. Run the shell script to provide access to the kernel headers. This may take a few minutes.

    $ sh installheaders.sh

11. Run the shell script **test.sh** to verify that the headers were successfully initialized.

    $ sh test.sh

```
root@edison:~/headers# sh installheaders.sh
root@edison:~/headers# sh test.sh
Documentation   arch     firmware  ipc     net       sound
Kconfig         block    fs        kernel  samples   tools
Makefile        crypto   include   lib     scripts   usr
Module.symvers  drivers  init      mm      security  virt
root@edison:~/headers#
```

Figure 3: Output from test.sh when installheaders.sh successfully initialized the headers required for kernel module development

Kernel Modules – intro_module.c

In this section, users will build, execute and observe the operation of a module that uses a few basic features of the Linux OS.

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

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

2. Download the **.zip archive** labelled **intro_module.zip**.
3. Extract the contents of **intro_module.zip** to a folder on the personal computer.
4. 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***.
5. Create a directory labelled **intro_module** and navigate to it.

   $ mkdir ~/intro_module
   $ cd ~/intro_module

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

   $ cd ~/intro_module
   $ ls

Figure 4: Files present in the 'intro_module' directory

9. Inspect the file labelled **intro_module.c**.

   $ cat intro_module.c

   Notice how the C-code source file does not have a **main** function. Instead, there are two core functions labelled **module_init()** and **module_exit()**. These functions form the basic structure of loadable kernel module in Linux. The kernel module will call the function passed to the function **module_init()** on initialization. Similarly, the kernel module will call the function passed to the function **module_exit()** on removal.

In the function **intro_init()** present in the C-code source file **intro_module.c**, one can observe the **printk()** function. This function writes the formatted string to the kernel message buffer. This is the Linux kernel equivalent of the **printf()** function and prints to the kernel message buffer instead of standard output. Thus, when the module is compiled and inserted into the kernel, the initialization function is called and this prints the input message to the kernel message buffer.

Furthermore, the kernel has access to functions such as **get_cpu()** that returns the ID of the CPU that is currently running the process. For correct operation, make sure to follow any **get_cpu()** call with a **put_cpu()** call. For more information, refer to the following link.

http://www.makelinux.net/books/lkd2/ch09lev1sec9

10. Issue the commands listed below.

    $ make clean
    $ make

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

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

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

11. Issue the command below to inspect the list of kernel modules currently loaded.

$ lsmod

```
root@edison:~/intro_module# 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:~/intro_module#
```

Figure 6: List of loadable kernel modules that are currently loaded

12. Insert the kernel module **intro_module.ko** by issuing the **insmod** command. Check the list of kernel modules.

$ insmod intro_module.ko
$ lsmod

```
root@edison:~/intro_module# insmod intro_module.ko
root@edison:~/intro_module# lsmod
Module                  Size  Used by
intro_module           12467  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
root@edison:~/intro_module#
```

Figure 7: Output from lsmod showing that 'intro_module' is currently loaded

13. Inspect the message buffer of the kernel.

$ dmesg

```
[157388.053349] Hello! | Timestamp: 1478800457.003355 | CPU: 1
```

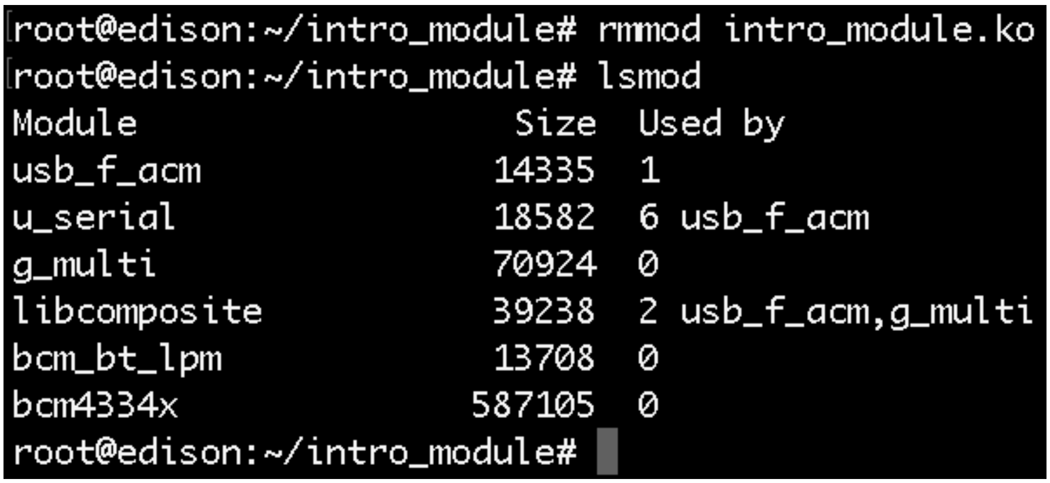
Figure 8: Output from dmesg showing that the kernel module functioned as expected

14. Issue the command below to clear the message buffer of the kernel

$ dmesg --clear

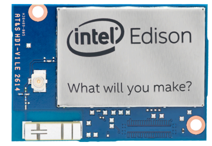
15. Remove the kernel module by issuing the following command.

$ rmmod intro_module
$ lsmod



```
root@edison:~/intro_module# rmmod intro_module.ko
root@edison:~/intro_module# 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:~/intro_module#
```

Figure 9: Output from lsmod after hello has been stopped



Tasks

Complete the steps below to gain a deeper understanding of kernel modules.

1. Modify **intro_module.c** such that the kernel module **intro_module.ko** will print

   "Goodbye! | Timestamp: ???\n"

   to the message buffer of the kernel on **exit**.

2. Build and insert the module.
3. Inspect the message buffer of the kernel.
4. Take a screenshot showing the messages

   "Hello! | Timestamp: ??? | CPU: ???\n"

   and

   "Goodbye! | Timestamp: ??? | CPU: ???\n"

   in the message buffer of the kernel.

5. Remove the kernel module.

Extension Tasks

Adding Arguments

Modify the **intro_module.c** kernel module to print a custom initialization message to the kernel message buffer instead of the **"Hello! | Timestamp: ??? | CPU: ???\n"** message. The message will be specified by the user as a command-line argument to the kernel module. There are details and example C code in Section 2.6 of the Linux Kernel Programming Guide.

http://www.tldp.org/LDP/lkmpg/2.6/html/lkmpg.html#AEN323