**Linux Kernel\_Project 1\_Report**

**Group members:**

* **1751063 Pham Bao Duy**
* **1751064 Nguyen Hoang Gia**

1. **Linux Kernel report**

The Linux kernel was conceived and created in 1991, based on UNIX kernel, by Linus Torvalds for his personal computer. Common components of Linux kernel:

1. Process Management: responsible for the following activities in connection with process management
   * Process creation and deletion.
   * Process suspension and resumption.
   * Provision of mechanisms for:
     + Process synchronization
     + Process communication
2. Memory Management: responsible for the following activities in connections with memory management
   * Keep track of which parts of memory are currently being used and by whom.
   * Decide which processes to load when memory space becomes available.
   * Allocate and deallocate memory space as needed.
3. Device Management: responsible for managing all the hardware devices
   * Manage all the input and output devices.
   * Keep track of the status of all the devices.
   * Exchange data with all devices
4. File System Management: the process of manipulating files, it management includes the process of creating, modifying and deleting the files.
5. Networking Management: responsible for managing packets according to TCP/IP model.
6. System Call interface: is the denomination for the entirety of all implemented and available system calls in a kernel.

The source code for the Linux kernel includes some directories:

* /arch: The /arch subdirectory contains all the architecture specific kernel code. It has further subdirectories, one per supported architecture, for example: x86, alpha, arm, mips, mk68, powerpc, sparc,…
* /drivers: All of the system's device drivers live in this directory. They are further sub-divided into classes of device driver, for example: block, char, pci, scsi,…
* /kernel: The main kernel code includes scheduler, handling code, data structure.
* /fs: All of the file system code. This is further sub-divided into directories, one per supported file system, for example: vfat, ext2,…
* /ipc: This directory contains the kernels inter-process communications code.
* /net: The kernel's networking code for managing packets according to TCP/IP model.
* /mm: This directory contains all of the memory management code
* …

Commonly concept:

* **User space** and **Kernel space**:
  + Kernel space is where the kernel executes and provides its services.
  + User space is that portion of system memory in which user processes run.
* **User mode** and **Kernel mode**:
  + In **Kernel mode**, the executing code has complete and unrestricted access to the underlying hardware. It can execute any CPU instruction and reference any memory address. Kernel mode is generally reserved for the lowest-level, most trusted functions of the operating system.
  + In **User mode**, the executing code has no ability to directly access hardware or reference memory. Code running in user mode must delegate to system APIs to access hardware or memory.
* **System Call** and **Interrupt**:
  + **System Call**: A request from an application program for the operating system to perform some hardware action on behalf of the application. System calls are initiated with a software interrupt assembly language instruction.
  + **Interrupt**: Signals sent to the CPU to tell the CPU to stop its current activities and execute the appropriate part of the OS. Allow OS to gain the control.
* **Process context** and **interrupt context:**
  + **Process context:** 
    - (Above process) A process which refers to the switching from User mode to Kernel mode is CPU registers need to be saved when the User mode process states and the contents on the stack, i.e. save process context of the current process, so as to perform again the when the process of switching state can be restored, it continues.
    - (The following process) It refers to a program executed after switching to Kernel mode, i.e., a process running in Kernel space portion.
  + **Interrupt context:**
    - (Above interrupt) Hardware interrupt triggered by a signal, resulting in a Kernel interrupt handler was called into Kernel space. This process, some of the variables and parameters of hardware is also passed to the Kernel, Kernel interrupt handling by these parameters. Hardware interrupt that can be seen above the pass parameters and the kernel need to save some other environment (mainly current interrupted process environment).
    - (Interrupted by the following) execution in Kernel space interrupt service routine.

1. **Code report**

Basically, when implementing a module for kernel, we must have 2 compulsory functions: init, exit. The frame of random\_module.c, basic:

|  |
| --- |
| #include <linux/init.h>  #include <linux/module.h>  static int \_\_init init\_project(void)  {  printk("RndNum initialized successfully.\n");  return 0;  }  static void \_\_exit exit\_project(void)  {  printk("RndNum removed and un-plugged successfully.\n");  }  module\_init(init\_project);  module\_exit(exit\_project);  MODULE\_LICENSE(“GPL”); |

Next, to communicate between Kernel Space and User Space, we need to implement some function for device file, says:

1. static int device\_open(struct inode \*, struct file \*);
2. static int device\_release(struct inode \*, struct file \*);
3. static ssize\_t device\_read(struct file \*, char \*, size\_t, loff\_t \*);

The (1) function will be called whenever in User space access to device (or open).

The (3) function will be called whenever device is close in user space.

The (2) function is when user space read into the device. This is when we generate the random number through the function get\_random\_bytes(&rand, sizeof(rand));.

**To run the Project:**

* Ubuntu / Linux
* Download or clone the project.
* Install essential components by running:

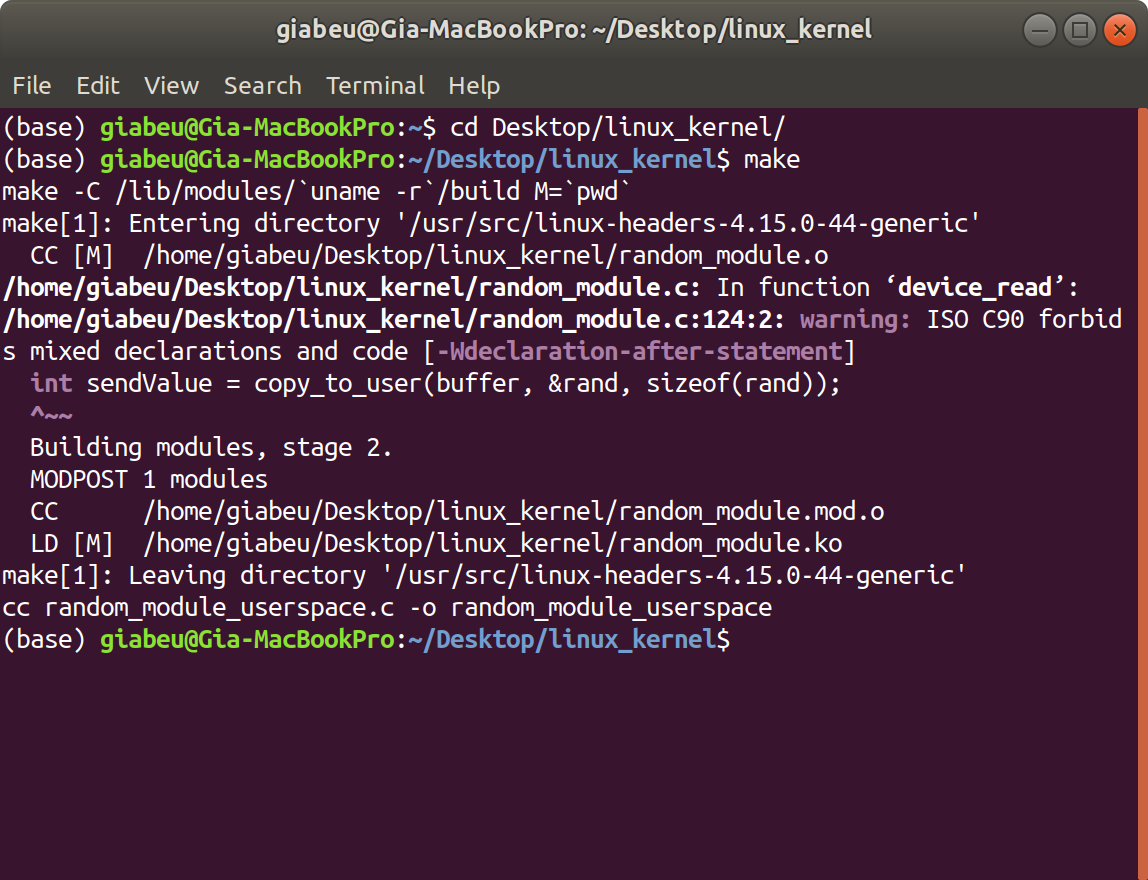
sudo apt-get install build-essential

* Change directory to the project folder.

**Step 1: Compile module in Kernel Space and User Space**

* Simply by running

make

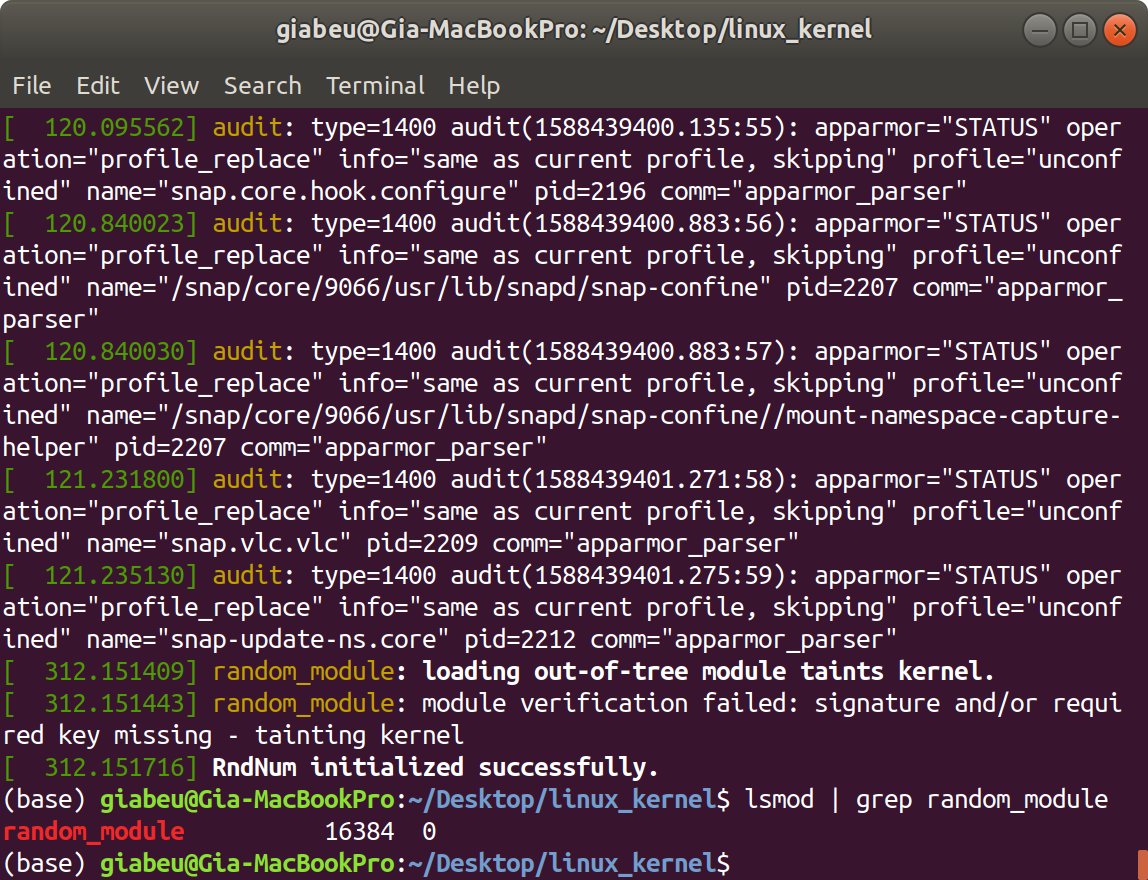
****

**Step 2: Insert module into the kernel & grant permission**

* Run

sudo insmod random\_module.ko

lsmod | grep random\_module



* Next, run

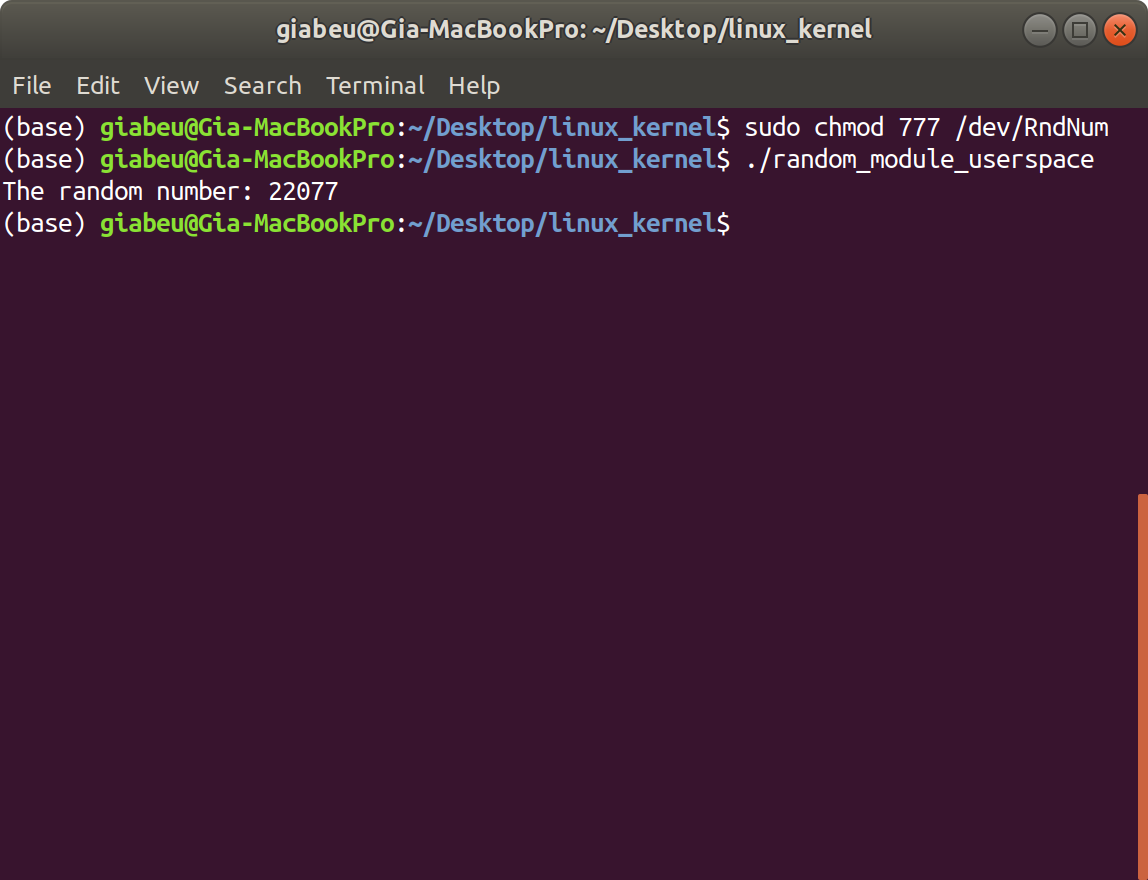
sudo chmod 777 /dev/RndNum

**Step 3: Get random number from user space**

* Run this command

./random\_module\_userspace

* Check the output, it will be some random number.



**Step 4: Remove module from the kernel and clean up**

* First, run

sudo rmmod random\_module.ko

* Then

make clean

