REPORT

Group information
Nguyen Ho Huu Nghia 1751013 - **17APCS2**
Dao Hieu 1751005 - **17APCS1**
Nguyen Truong Vinh Thuyen 1751042 - **17APCS1**

Note: The sections concerning the implementation of our kernel module are at section 4 and 5.

1. **Linux kernel**
   Linux kernel, commonly also called Linux. It is an Unix-like operating system kernel. Many computer systems are deployed using the Linux kernel from PCs, embedded systems, supercomputers and also mobile devices(Android).
   a. **Linux kernel**: include 6 parts:
      - **Process management**:
        - Process representation: The process is represented in a structure called *task_struct.* The structure contains all information about the thread, such as its id, flags, state, relationship with other threads, etc.
        - Most processes are created and represented by a dynamically loaded *task_struct*, except for the *init* process, which is always loaded and represented by a statically loaded *task_struct.*
        - Processes are collected in either a hash table (the key is the process id) or a circular linked list.
        - Process creation: creating threads in either the user-space or the kernel seems different, as they use different functions: fork() and kernel_thread(). However, fork() is going to trigger a system call that calls the sys_fork() function, which is going to eventually call do_fork(), and kernel_thread() calls do_fork() too.
        - Process scheduling: the Linux scheduler keeps a set of lists of priority level, each list keeps the references to the task_struct instances of the processes.
        - Process destruction: basically, a process exits when there is some fatal signal, or when the process calls exit(). In both of these cases, the function do_exit() in the kernel is eventually called. The destruction process involves setting its flag so that no other parts of the kernel does not reference it, releasing the resources and making notification about the process exit.
      - **Memory management**: this subsystem, whose source code is in the directory /mm, includes Linux's implementation of:
        - Memory allocation
        - Address mapping

        - Virtual memory
        - Demand paging
    - **Device management**: Linux let processes interact with devices as if they are files. All devices are stored as node files in the /dev directory. The device management system is also responsible for keeping tracks of all devices, handling processes' requests for devices, allocating and deallocating the resources for devices.
    - **Filesystem management**
        - Function: Provide a virtual file system, an interface to help users to access data and modify data. Manage stored data in hardware.
        - Directory structure: Filesystem hierarchy standard

Directory	**Description**
/ Root filesystem	The top-level directory of the filesystem Contain every file that is required to boot the Linux system.
/bin	This directory contains executable files
/boot	Contain bootloader, kernel executable and configuration files that are needed to boot the Linux system
/home	Store files and documents of users. Each user has his/her own subdirectory.
/root	Home directory for the root user
/dev	Contain device files for hardware devices.
/etc	Contain files of local system configuration for the host computer
/lib	Contain shared library files that are needed to boot the system
/opt	Contain optional files

/tmp	Store temporary files
/usr	Contain
/var	Store variable files.
/mnt	Temporary mount point
/media	Also a temporary mount point

- **Networking management**: managing network packets with TCP/IP.
- **System call Interface**: provide processes with a way to use OS services through system calls.

b. **Some directories in Linux open-source**:

Directory	Used for
/arch	All the architecture-specific code is in this directory and in the include/asm-<arch> directories.
/block	Contains source code that deploys the scheduling task for storage devices.
/drivers	As a general rule, code to run peripheral devices is found in subdirectories of this directory.
/fs	Both the generic filesystems code (known as the VFS, or Virtual File System) and the code for each different filesystems are found in this directory.
/ipc	"IPC" stands for "Inter-Process Communication". It contains the code for shared memory, semaphores, and other forms of IPC.
/kernel	Generic kernel level code that doesn't fit anywhere else goes in here. The upper level system call code is here, along with the printk() code, the scheduler, signal handling code, and much more.
/mm	High level memory management code is in this directory.

/net	The high-level networking code is here.

c. **Some definition**:
    - **User-space and kernel space**: Memory is divided into two distinct areas:
        - User-space: which is a set of locations where normal user processes run (i.e everything other than the kernel). The role of the kernel is to manage applications running in this space from messing with each other, and the machine.
        - Kernel space: which is the location where the code of the kernel is stored, and executes under.
    - **User mode and kernel mode**: Hardware provides at least two modes (mode bit in CPU):
        - User mode: mode bit = 1
            - Can only execute a subset of instructions
            - Can only reference a subset of memory locations
        - Kernel mode: mode bit = 0
            - Can execute all machine instructions
            - Can reference all memory locations
    - **System calls and interrupts**
        - System calls: A mechanism that provides the interface between a process and the OS to help a program request services from the kernel. When a process in user mode requires a service that is provided by the OS, for example, process creation, process management, I/O device handling, etc.
        - Interrupts: A signal to CPU indicating that is an event that needs immediate care about. That requests the CPU to interrupt the currently executing. The process will be saved its state and suspended current activity if the request is accepted.
    - **Process context and interrupt context**
        - Process context: When the CPU is processing to respond to a system call request.
        - Interrupt context: When the CPU is processing to respond to an interrupt signal.

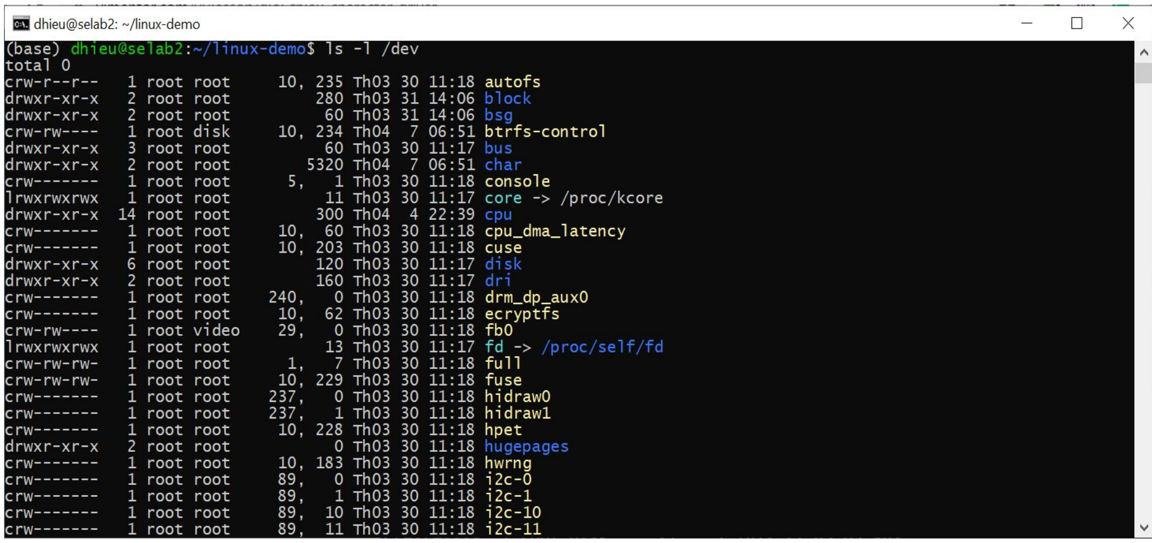
2. Linux kernel modules:

a. Linux kernel module
- Linux kernel module or Loadable kernel module is an object file containing code which can be loaded or unloaded into the kernel upon demand. These modules are typically used to add support for new hardware (as device drivers) and/or file systems, or for adding system calls.
- Device drivers are a type of module that deals with hardware devices.

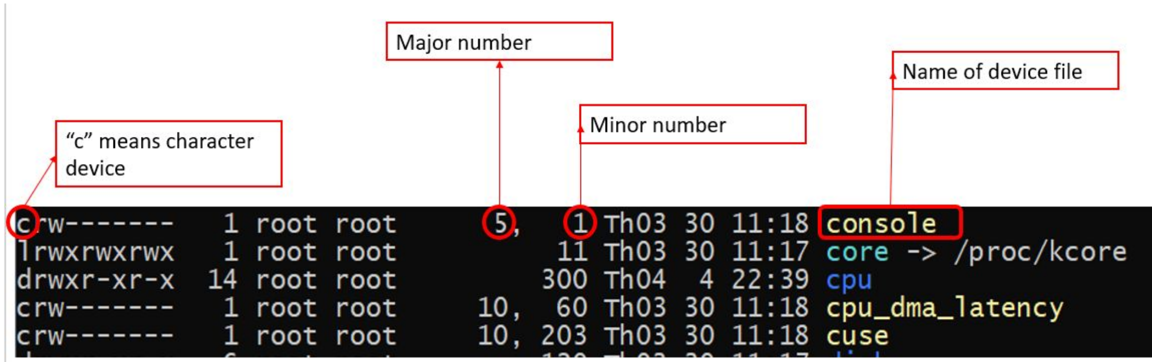
b. **Advantages**:

        - Without the Linux kernel modules, all possible anticipated functionality already compiled have to be loaded directly into the kernel. This leads to a huge memory of the kernel not being used → waste memory. Hence, the Linux kernel modules can save the memory for the kernel.
        - When adding new functionalities, the OS doesn't need to rebuild or reboot the kernel
    c. **Write a module**:
        - Step 1:
            - Create file <module_name>.c
            - Create Makefile
            - Create Kbuild
        - Step 2:
            - Compile module → <module_name>.ko
        - Step 3:
            - Insert a module into the kernel:
                - sudo insmod <module_name>.ko
            - Check inserted module:
                - lsmod | grep <module_name>
            - Print the kernel buffer:
                - dmesg
            - Remove module from the kernel:
                - sudo rmmod <module_name>.ko

3. **Character device driver**
    a. **Device file**
        - In Linux, there are six types of file: regular file, directory, pipe, socket, symbolic link and device file.
        - Linux treats the devices as files called device files. So when a process reads or writes on a device, it reads/writes on the device file that links to this device as a regular file.
    b. **How a process reads/writes on a device**:
        - Process interacts with the device file through system call or library call in an indirect way.
        - Kernel invokes the entry point of the device driver corresponding to the device file. Each entry point is a function of the device driver.
        - Through the device driver, the CPU can communicate with the corresponding device.
    c. **Device number**
        - The kernel manages the device files through device numbers:
            - Major number: used to distinguish between devices.
            - Minor number: used to distinguish between different drivers for a device.
        - We can see a list of all devices by using "ls -l /dev" on terminal.

■ 

```
(base) dhieu@selab2:~/linux-demo$ ls -l /dev
total 0
crw-r--r--   1 root root     10, 235 Th03 30 11:18 autofs
drwxr-xr-x   2 root root         280 Th03 31 14:06 block
drwxr-xr-x   2 root root          60 Th03 31 14:06 bsg
crw-rw----   1 root disk     10, 234 Th04  7 06:51 btrfs-control
drwxr-xr-x   3 root root          60 Th03 30 11:17 bus
drwxr-xr-x   2 root root        5320 Th04  7 06:51 char
crw-------   1 root root      5,   1 Th03 30 11:18 console
lrwxrwxrwx   1 root root          11 Th03 30 11:17 core -> /proc/kcore
drwxr-xr-x  14 root root         300 Th04  4 22:39 cpu
crw-------   1 root root     10,  60 Th03 30 11:18 cpu_dma_latency
crw-------   1 root root     10, 203 Th03 30 11:18 cuse
drwxr-xr-x   6 root root         120 Th03 30 11:17 disk
drwxr-xr-x   2 root root         160 Th03 30 11:17 dri
crw-------   1 root root    240,   0 Th03 30 11:18 drm_dp_aux0
crw-------   1 root root     10,  62 Th03 30 11:18 ecryptfs
crw-rw----   1 root video    29,   0 Th03 30 11:18 fb0
lrwxrwxrwx   1 root root          13 Th03 30 11:17 fd -> /proc/self/fd
crw-rw-rw-   1 root root      1,   7 Th03 30 11:18 full
crw-rw-rw-   1 root root     10, 229 Th03 30 11:18 fuse
crw-------   1 root root    237,   0 Th03 30 11:18 hidraw0
crw-------   1 root root    237,   1 Th03 30 11:18 hidraw1
crw-------   1 root root     10, 228 Th03 30 11:18 hpet
drwxr-xr-x   2 root root           0 Th03 30 11:18 hugepages
crw-------   1 root root     10, 183 Th03 30 11:18 hwrng
crw-------   1 root root     89,   0 Th03 30 11:18 i2c-0
crw-------   1 root root     89,   1 Th03 30 11:18 i2c-1
crw-------   1 root root     89,  10 Th03 30 11:18 i2c-10
crw-------   1 root root     89,  11 Th03 30 11:18 i2c-11
```

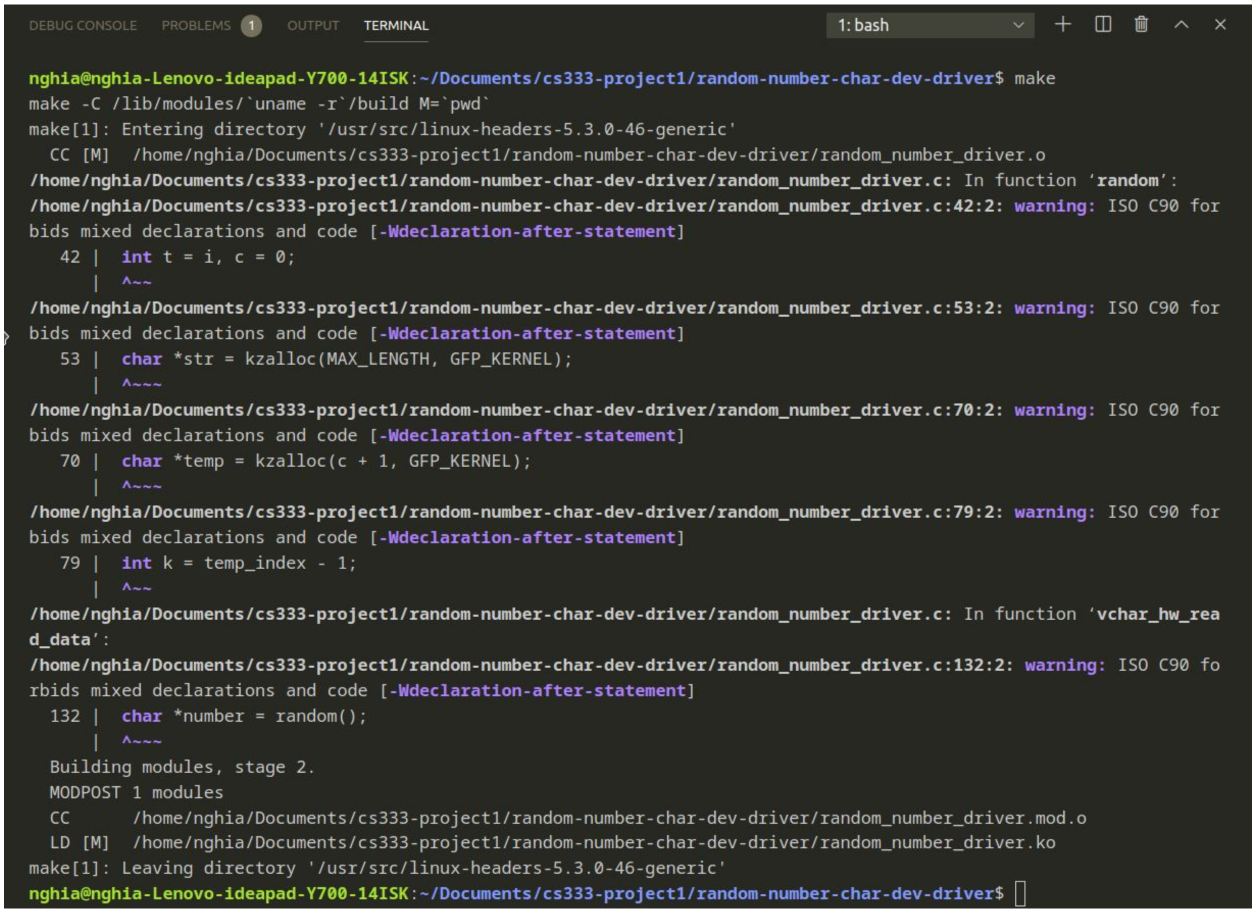
■ 

```
crw-------   1 root root      5,   1 Th03 30 11:18 console
lrwxrwxrwx   1 root root          11 Th03 30 11:17 core -> /proc/kcore
drwxr-xr-x  14 root root         300 Th04  4 22:39 cpu
crw-------   1 root root     10,  60 Th03 30 11:18 cpu_dma_latency
crw-------   1 root root     10, 203 Th03 30 11:18 cuse
```

d. **Character device driver:** Include two parts:
   - **OS-specific**:
     - init function.
     - exit function.
     - Entry point functions: open(), release(), read(), write(), ...
   - **Device-specific**:
     - Device initialization / release.
     - Read/write into device registers.
     - Interrupt function.

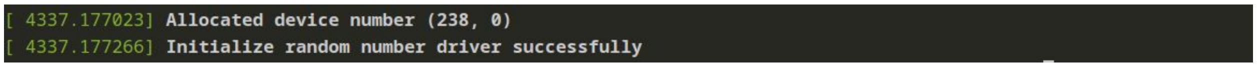
4. How to use

a. Compile and load the module to the kernel
   - cd ./random-number-char-dev-driver
   - make all



```
nghia@nghia-Lenovo-ideapad-Y700-14ISK:~/Documents/cs333-project1/random-number-char-dev-driver$ make
make -C /lib/modules/`uname -r`/build M=`pwd`
make[1]: Entering directory '/usr/src/linux-headers-5.3.0-46-generic'
  CC [M]  /home/nghia/Documents/cs333-project1/random-number-char-dev-driver/random_number_driver.o
/home/nghia/Documents/cs333-project1/random-number-char-dev-driver/random_number_driver.c: In function ‘random’:
/home/nghia/Documents/cs333-project1/random-number-char-dev-driver/random_number_driver.c:42:2: warning: ISO C90 for
bids mixed declarations and code [-Wdeclaration-after-statement]
   42 |  int t = i, c = 0;
      |  ^~~
/home/nghia/Documents/cs333-project1/random-number-char-dev-driver/random_number_driver.c:53:2: warning: ISO C90 for
bids mixed declarations and code [-Wdeclaration-after-statement]
   53 |  char *str = kzalloc(MAX_LENGTH, GFP_KERNEL);
      |  ^~~~
/home/nghia/Documents/cs333-project1/random-number-char-dev-driver/random_number_driver.c:70:2: warning: ISO C90 for
bids mixed declarations and code [-Wdeclaration-after-statement]
   70 |  char *temp = kzalloc(c + 1, GFP_KERNEL);
      |  ^~~~
/home/nghia/Documents/cs333-project1/random-number-char-dev-driver/random_number_driver.c:79:2: warning: ISO C90 for
bids mixed declarations and code [-Wdeclaration-after-statement]
   79 |  int k = temp_index - 1;
      |  ^~~
/home/nghia/Documents/cs333-project1/random-number-char-dev-driver/random_number_driver.c: In function ‘vchar_hw_rea
d_data’:
/home/nghia/Documents/cs333-project1/random-number-char-dev-driver/random_number_driver.c:132:2: warning: ISO C90 fo
rbids mixed declarations and code [-Wdeclaration-after-statement]
  132 |  char *number = random();
      |  ^~~~
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/nghia/Documents/cs333-project1/random-number-char-dev-driver/random_number_driver.mod.o
  LD [M]  /home/nghia/Documents/cs333-project1/random-number-char-dev-driver/random_number_driver.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.3.0-46-generic'
nghia@nghia-Lenovo-ideapad-Y700-14ISK:~/Documents/cs333-project1/random-number-char-dev-driver$
```

- sudo rmmod random_number_driver
  - # remove if loaded
- sudo insmod random_number_driver.ko
- dmesg
  - # see if the module is loaded

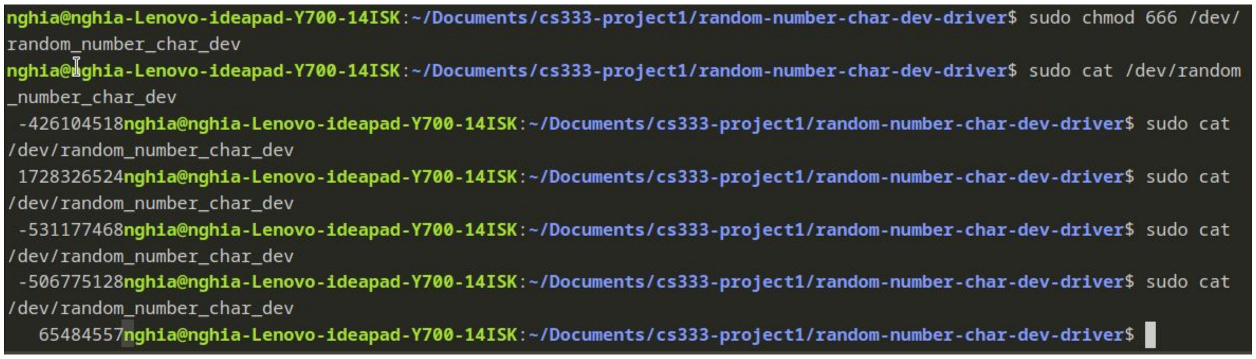


```
[ 4337.177023] Allocated device number (238, 0)
[ 4337.177266] Initialize random number driver successfully
```

- sudo chmod 666 /dev/random_number_char_dev

b. Everytime, you read the file, a new random number is returned
- The string returned has 11 characters, if the random number does not take up 11 characters, space is padded at the front.

c. An example way to run is through `cat`
- sudo cat /dev/random_number_char_dev



```
nghia@nghia-Lenovo-ideapad-Y700-14ISK:~/Documents/cs333-project1/random-number-char-dev-driver$ sudo chmod 666 /dev/
random_number_char_dev
nghia@nghia-Lenovo-ideapad-Y700-14ISK:~/Documents/cs333-project1/random-number-char-dev-driver$ sudo cat /dev/random
_number_char_dev
 -426104518nghia@nghia-Lenovo-ideapad-Y700-14ISK:~/Documents/cs333-project1/random-number-char-dev-driver$ sudo cat
/dev/random_number_char_dev
 1728326524nghia@nghia-Lenovo-ideapad-Y700-14ISK:~/Documents/cs333-project1/random-number-char-dev-driver$ sudo cat
/dev/random_number_char_dev
 -531177468nghia@nghia-Lenovo-ideapad-Y700-14ISK:~/Documents/cs333-project1/random-number-char-dev-driver$ sudo cat
/dev/random_number_char_dev
 -506775128nghia@nghia-Lenovo-ideapad-Y700-14ISK:~/Documents/cs333-project1/random-number-char-dev-driver$ sudo cat
/dev/random_number_char_dev
   65484557nghia@nghia-Lenovo-ideapad-Y700-14ISK:~/Documents/cs333-project1/random-number-char-dev-driver$
```

5. **The purpose of each function/struct:**

Function/Structure	Purpose

typedef struct vchar_dev {
unsigned char *control_registers;
unsigned char *status_registers;
unsigned char *data_registers;
} vchar_device_t;
	The class for the character device

struct vchar_driver {
dev_t device_number; // device number consist of major and minor number
struct class *device_class; // the device class
struct device *device; // the device
vchar_device_t *vchar_hardware; // the pointer to the underlying character device struct
struct cdev *vcdev; // the pointer to the character device (cdev is supplied by the library)
unsigned int open_cnt; // the number of open entry points
};
	The main class for the driver

char *random(void)	Generate a random number as a char array with the help from get_random_bytes()
int vchar_hw_init(vchar_device_t *hardware)	Initialize the character device
void vchar_hw_exit(vchar_device_t *hardware)	What to do when the device is destroyed

int vchar_hw_read_data(vchar_device_t *hw, int start_register, int num_registers, char *kbuf)	Read the data from the device's registers, put that in the buffer and then return the number of bytes read
int vchar_hw_write_data(vchar_device_t *hw, int start_register, int num_registers, char *kbuf)	Write the data from the kernel buffer to registers, then return the number bytes written
static int vchar_driver_open(struct inode *inode, struct file *filp)	Entry point function for open operation
static int vchar_driver_release(struct inode *inode, struct file *filp)	Entry point function for release operation
static ssize_t vchar_driver_read(struct file *filp, char __user *user_buffer, size_t len, loff_t *off)	Entry point function for read operation
static ssize_t vchar_driver_write(struct file *filp, const char __user *user_buffer, size_t len, loff_t *off)	Entry point function for write operation
static int __init vchar_driver_init(void)	Initializing the driver
static void __exit vchar_driver_exit(void)	Unregistering and freeing resources when the driver exits