Activity 11 : Kernel Module
Instructor: Krerk Piromsopa, Ph. D.

1. Dummy Driver. Create a dummy kernel module (dummy.c) and use the given Makefile to build the module.

```
// dummy.c
#include <linux/module.h>       /* Needed by all modules */
#include <linux/kernel.h>       /* Needed for KERN_INFO */
MODULE_LICENSE("GPL");
MODULE_AUTHOR("KRERK PIROMSOPA, PH.D. <Krerk.P@chula.ac.th>");
MODULE_DESCRIPTION("\"cpmod\" Dummy Kernel Module");
int init_module(void)
{
        printk(KERN_INFO "CPMOD: init\n");
         /* * non 0 - means init_module failed
                        / module can't be loaded.
*/ return 0;
}
void cleanup_module(void)
{
      printk(KERN_INFO "CPMOD: cleanup\n");
}
```

Makefile

```
obj-m += dummy.o
all:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Check Point #1

Use *insmod*, *rmmod* to install/remove module. You have to demonstrate that the module has been installed and removed (using *dmesg* and *lsmod*).

2. **Simple character device driver.** In this exercise, you have to compile the given character driver

module (osinfo).

```
// osinfo.c
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <asm/uaccess.h>
/* Needed by all modules */
/* Needed for KERN_INFO */
MODULE_LICENSE("GPL");
MODULE_AUTHOR("KRERK PIROMSOPA, PH.D. <Krerk.P@chula.ac.th>");
MODULE_DESCRIPTION("\"osinfo\" Character Device");
#define DEVICENAME "osinfo"
static int dev_major;
static int dev_open =0;
static char *f_ptr;
static const char f_data0[] = "0:CP ENG CU OS 2022S2 - Instructors\n1:\tVeera Muangsin,
Ph.D.\n2:\tKrerk Piromsopa, Ph.D.\n3:\tThongchai Rojkangsadan\n";
// prototypes for device functions
static int device_open(struct inode *, struct file *);
static int device_release(struct inode *inode, struct file *file);
static ssize_t device_read(struct file *, char *, size_t, loff_t *);
// File operations structor
// Only implement those that will be used.
static struct file_operations dev_fops = {
        .read = device_read,
        .open = device_open,
        .release = device_release
};
int init_module(void)
{
printk(KERN_INFO "CPCHAR: dev osinfo init\n");
dev_major=register_chrdev(0, DEVICENAME, &dev_fops);
if (dev_major < 0) {
        printk(KERN_ALERT "Fail register_chrdev osinfo with %d\n",dev_major);
        return dev_major;
}
printk(KERN_INFO "Device MajorNumber %d.\n", dev_major);
printk(KERN_INFO "To create a device file:\n");
printk(KERN_INFO "\t'mknod /dev/%s c %d 0'.\n", DEVICENAME, dev_major);
printk(KERN_INFO "Try varying minor numbers.\n");
printk(KERN_INFO "Please remove the device file and module when done.\n");
 /* * non 0 - means init_module failed   */
return 0;
}

void cleanup_module(void)
{
        printk(KERN_INFO "CPCHAR: dev osinfo cleanup\n");
        unregister_chrdev(dev_major, DEVICENAME);
}
static int device_open(struct inode *inode, struct file *file)
{
        if (dev_open)
                return -EBUSY;
```

```
        dev_open++;
        printk(KERN_INFO "dev minor %d\n", MINOR(inode->i_rdev));
        f_ptr=(char * )f_data0;
        // lock module
        try_module_get(THIS_MODULE);
        return 0;
}
static int device_release(struct inode *inode, struct file *file)
{
dev_open--;             /* We're now ready for our next caller */
// release module
module_put(THIS_MODULE);
return 0;
}
static ssize_t device_read(struct file *filp,
                           char *buffer,
/* see include/linux/fs.h   */
/* buffer to fill with data */
/* length of the buffer     */
size_t length,
loff_t * offset)
{
int bytes_read=0;
if (*f_ptr ==0) {
        return 0;
}
while (length && *f_ptr) {
        put_user(*(f_ptr++), buffer++);
        length--;
        bytes_read++;
}
return bytes_read;
}
```

Check Point #2

Use *insmod* to install the driver. Once the driver is installed, use the *mknod* command to create **/dev/osinfo** device file. You may obtain the major device number from *dmesg*. Please also show the content of /dev/osinfo.

3. Minor Device Modify the osinfo driver to display your name (student name) when the minor device number is 1. (Note that the system must display the instructor information when the minor device number is 0.)

This is the expected result.

```
krerk@OSBox:~/kmod/cpdev$ ls -al /dev/osinfo*
crw-r--r-- 1 root root 250, 0 Nov 23 06:51 /dev/osinfo
crw-r--r-- 1 root root 250, 1 Nov 23 06:51 /dev/osinfo1
krerk@OSBox:~/kmod/cpdev$ cat /dev/osinfo
0:CP ENG CU OS 2022S2 - Instructors
1:      Veera Muangsin, Ph.D.
2:      Krerk Piromsopa, Ph.D.
3:      Thongchai Rojkangsadan
krerk@OSBox:~/kmod/cpdev$ cat /dev/osinfo1
0:CP ENG CU OS 2022S2 – Students, Group Name: [groupname]
1:      4123456721 [member 1]
2:      4123456721 [member 2]
```

Check Point #3

Create a new device file (/dev/osinfo1). Show the expected result to your instructor. The content of /dev/osinfo1 must display your group information.

4. CPUInfo Device Driver. In this exercise, you are asked to design a character device driver that will show the vendor ID, features, and serial number of your processor.
The function for taking the vendor ID, features and serial number is provided.

```
static inline void native_cpuid(unsigned int *eax, unsigned int *ebx,
                                unsigned int *ecx, unsigned int *edx)
{
/* ecx is often an input as well as an output. */
asm volatile("cpuid"
    : "=a" (*eax),
      "=b" (*ebx),
      "=c" (*ecx),
      "=d" (*edx)
    : "0" (*eax), "2" (*ecx)
    : "memory");
}
int main(int argc, char **argv) {
// Code snippet
unsigned eax, ebx, ecx, edx;
// for obtaining the features
eax = 0; /* processor info and feature bits */
native_cpuid(&eax, &ebx, &ecx, &edx);
printf("Vendor ID ");
printf("%c%c%c%c", (ebx) & 0xFF, (ebx>>8) & 0xFF, (ebx>>16) & 0xFF, (ebx>>24) &
0xFF);
printf("%c%c%c%c", (edx) & 0xFF, (edx>>8) & 0xFF, (edx>>16) & 0xFF, (edx>>24) &
0xFF);
printf("%c%c%c%c", (ecx) & 0xFF, (ecx>>8) & 0xFF, (ecx>>16) & 0xFF, (ecx>>24) &
0xFF);
printf("\n");
// for obtaining the features
eax = 1; /* processor info and feature bits */
native_cpuid(&eax, &ebx, &ecx, &edx);
printf("stepping %d\n", eax & 0xF);
printf("model %d\n", (eax >> 4) & 0xF);
printf("family %d\n", (eax >> 8) & 0xF);
printf("processor type %d\n", (eax >> 12) & 0x3);
printf("extended model %d\n", (eax >> 16) & 0xF);
printf("extended family %d\n", (eax >> 20) & 0xFF);
// for obtaining the serial number
eax = 3; /* processor serial number */
native_cpuid(&eax, &ebx, &ecx, &edx);
printf("serial number 0x%08x%08x\n", edx, ecx);
// For more details, see https://en.wikipedia.org/wiki/CPUID
}
```

(**Note** that printf cannot be used within the kernel. You have to convert the code into a character device driver.)

Check Point #4

Create the /dev/cpuinfo device file. The content must display the CPUID of your processor.

5. SysInfo Driver. Create a character device (cpsysinfo) as a kernel module. This character device must provide information about your operating systems as follows:

Minor 0: active processes on your operating system.

Minor 1: Amount of memory

Hint:

Given that a kernel module is a part of the kernel, we can simply access related data structures within the kernel for any information. To browse the kernel source, visit http://www.kernel.org/ and select your kernel version.

The code for accessing task structure is provided.

```
#include <linux/sched.h>
/* sample code */
struct task_struct *task;
char buff[1000];

for_each_process(task) {
     snprintf(buff,1000,"%d,%s \n", task->pid, task->comm);
```

(For more information see https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/include/linux/sched.h)

To get information about the memory, see (https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/fs/proc/meminfo.c)

Check Point #5

Create the /dev/cp-psinfo device file (Minor 0). The file must provide pid, command of all processes. The output should look like this.

```
1,systemd
2,kthreadd
3,ksoftirqd/0
5,kworker/0:0H
8,rcu_sched
9,rcu_bh

…..
```

Check Point #6

Create the /dev/cp-meminfo device file (Minor 1). The file must provide the total amount of available memory. (The result should look similar to /proc/meminfo.)

```
MemTotal:       65906740 kB
MemFree:          272508 kB
MemAvailable:   63415376 kB
```