Kernel Programming
Assignment 2

Aurobindo Mondal
143050082

January 15, 2016

Simple Loadable Kernel Module :

Program Code :

```
#include <linux/module.h> /* Needed by all modules */
#include <linux/kernel.h> /* Needed for KERN_INFO */
int init_module(void)
{
  printk(KERN_INFO "Hello world 1.\n");
  return 0;
}
void cleanup_module(void)
{
  printk(KERN_INFO "Goodbye world 1.\n");
}
```

hello–1.c

Compile and Run :

Makefile :

```
  obj-m += hello-1.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
```

To compile, run the command `make`. After the compilation, a kernel object file named *hello-1.ko* is created.

Insert the module into the Kernel

For inserting the module into the kernel, we require the command `sudo insmod hello-1.ko`

As we insert the module, the `init_module()` function is called. The function prints the output message in the system log as the priority set for printk() is KERN_INFO. The function prints the output message in the system log as the priority set for printk() is KERN_INFO.

Remove the module into the Kernel

For removing the module into the kernel, we require the command `sudo rmmod hello_1` As we remove the module from the kernel, the `cleanup_module()` function is called. The function prints the output message in the system log as the priority set for printk() is KERN_INFO.

printk() :

`printk()` is a logging mechanism for the kernel. Each `printk()` statement comes with a priority, as follows :

- **KERN_EMERG :** Used for emergency messages.

- **KERN_ALERT :** A situation requiring immediate action. Output printed on the console.
- **KERN_CRIT :** Critical conditions, often related to serious hardware or software failures.
- **KERN_ERR :** Used to report error conditions.
- **KERN_WARNING :** Report Warnings.
- **KERN_NOTICE :** Bring into notice security-related conditions.
- **KERN_INFO :** Informational messages. Messages are stored in `/var/log/syslog`.
- **KERN_DEBUG :** Used for debugging messages.

Output from the module to the console :

Program Code :

```
#include <linux/module.h> /* Needed by all modules */
#include <linux/kernel.h> /* Needed for KERN_INFO */
int init_module(void)
{
  printk(KERN_ALERT "Hello world 1.\n");
  return 0;
}
void cleanup_module(void)
{
  printk(KERN_ALERT "Goodbye world 1.\n");
}
```

hello–2.c

KERN_ALERT causes to display the output on to the console.

Input from Command Line :

Program Code :

```
#include <linux/module.h> /* Needed by all modules */
#include <linux/kernel.h> /* Needed for KERN_INFO */
#include <linux/moduleparam.h>
#include <linux/init.h>

int arr_argc = 0;
int myintArrar[3] = {10, 20, 30};

module_param_array(myintArray, int, &arr_argc, 0000);
MODULE_PARM_DESC(myintArray, "An array of integers");
```

```
int init_module(void)
{
  int i;
  for (i = 0; i < (sizeof myintArray / sizeof
      (int)); i++)
  {
    printk(KERN_INFO "myintArray[%d] = %d\n", i,
        myintArray[i]);
  }
  return 0;
}
void cleanup_module(void)
{
  printk(KERN_INFO "Goodbye world 1.\n");
}
```

hello–3.c

Compile and Run :

Commands executed are :

```
make
sudo insmod hello-3.c myintArray=10,20,30
```

NOTE :

To view the messages output by the kernel module, we need to acces the `/var/log/syslog` or just type the command `dmesg`.