LINUX OPERATING SYSTEM
CS-1446
Assignment-2
Submitted by: Mrinmay Mukherjee
Scholar Id: 15-1-5-057

Writing a kernel module and Testing

Contents of /var/log/kern.log

```
Feb 26 22:58:58 CS-1443 NetworkManager[743]: <info>  [1551202138.3663]   gateway
192.168.44.1
Feb 26 22:58:58 CS-1443 NetworkManager[743]: <info>  [1551202138.3663]   server
identifier 192.168.47.224
Feb 26 22:58:58 CS-1443 NetworkManager[743]: <info>  [1551202138.3663]   lease
time 21600
Feb 26 22:58:58 CS-1443 NetworkManager[743]: <info>  [1551202138.3663]
nameserver '172.16.30.50'
Feb 26 22:58:58 CS-1443 NetworkManager[743]: <info>  [1551202138.3663]   domain
name 'cse.nits.ac.in'
Feb 26 22:58:58 CS-1443 NetworkManager[743]: <info>  [1551202138.3664] dhcp4
(eno1): state changed bound -> bound
Feb 26 23:34:22 CS-1443 kernel: [21026.698060] myname: loading out-of-tree
module taints kernel.
Feb 26 23:34:22 CS-1443 kernel: [21026.698090] myname: module verification
failed: signature and/or required key missing - tainting kernel
Feb 26 23:34:22 CS-1443 kernel: [21026.698385] My name is MRINMAY MUKHERJEE
Feb 26 23:35:02 CS-1443 kernel: [21066.659073] Exiting.... BYE!!
```

Code

```c
#include <linux/init.h>
#include <linux/module.h>
MODULE_LICENSE("GPL");

// Function to specify what happens when the module is loaded
static int myname_init(void)
{
```

```
    printk(KERN_ALERT "My name is MRINMAY MUKHERJEE\n"); //Include your name
    return 0;
}


// Function to specify what happens when the module is removed
static void myname_exit(void)
{
    printk(KERN_ALERT "Exiting.... BYE!! \n");
}

// Specifying the functions to call during the init and removal of module
module_init(myname_init);
module_exit(myname_exit);
```

Documentation

What is a kernel module?
Modules are pieces of code that can be loaded and unloaded into the kernel upon demand. They extend the functionality of the kernel without the need to reboot the system. For example, one type of module is the device driver, which allows the kernel to access hardware connected to the system. Without modules, we would have to build monolithic kernels and add new functionality directly into the kernel image. Besides having larger kernels, this has the disadvantage of requiring us to rebuild and reboot the kernel every time we want new functionality.

Functions and Macros used
Many parts of the kernel are well served as a module (dynamically-loadable parts of the kernel). Using the module_init() and module_exit() macros it is easy to write code without #ifdefs which can operate both as a module or built into the kernel.

The following functions/macros were used while writing the kernel module.

1. **__initcall()/module_init() include/linux/init.h** The module_init() macro defines which function is to be called at module insertion time (if the file is compiled as a module), or at boot time: if the file is not compiled as a module the module_init() macro becomes equivalent to __initcall(), which through linker magic ensures that the function is called on boot.

   The function can return a negative error number to cause module loading to fail (unfortunately, this has no effect if the module is compiled into the kernel). This function is called in user context with interrupts

enabled, so it can sleep.

2. **module_exit() include/linux/init.h** This macro defines the function to be called at module removal time (or never, in the case of the file compiled into the kernel). It will only be called if the module usage count has reached zero. This function can also sleep, but cannot fail: everything must be cleaned up by the time it returns.

   Note that this macro is optional: if it is not present, your module will not be removable (except for 'rmmod -f').

3. **printk() include/linux/kernel.h** printk() feeds kernel messages to the console, dmesg, and the syslog daemon. It is useful for debugging and reporting errors, and can be used inside interrupt context, but use with caution: a machine which has its console flooded with printk messages is unusable. It uses a format string mostly compatible with ANSI C printf, and C string concatenation to give it a first "priority" argument:

```
    printk(KERN_INFO "i = %u\n", i);
```

   See include/linux/kernel.h; for other KERN_ values; these are interpreted by syslog as the level. Special case: for printing an IP address use

```
    __be32 ipaddress;
    printk(KERN_INFO "my ip: %pI4\n", &ipaddress);
```

   printk() internally uses a 1K buffer and does not catch overruns

4. **MODULE_LICENCE** In kernel 2.4 and later, a mechanism was devised to identify code licensed under the GPL (and friends) so people can be warned that the code is non open-source. This is accomplished by the MODULE_LICENSE() macro. By setting the license to GPL, the "no license" warning will not be printed.

Makefile

```
ifneq ($(KERNELRELEASE),)
    obj-m := myname.o
else
    KERNELDIR ?= \
                /lib/modules/`uname -r`/build/
    PWD := `pwd`
```

```
default:
    $(MAKE) -C $(KERNELDIR) \
        M=$(PWD) modules
endif
clean:
    rm -f *.ko *.o Module* *mod*
```