Programming Project 1 – Linux Kernel Modules

Presented by
Felix Adrian Lucaciu
ID 27397941
Michael Mognabosco
ID 27189737

COEN-346
Section YK-X

Presented to:
Dr. Abdelazi Trabelsi

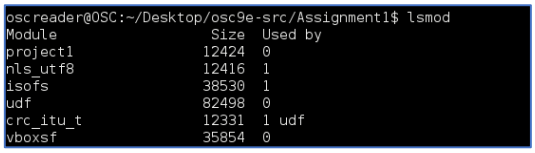
Due on: February 2nd 2018
Concordia University Department of Computer and Electrical Engineering

Introduction

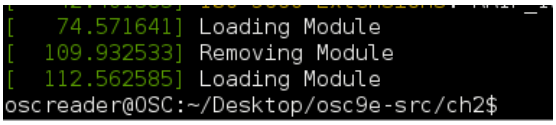
The goal of this project is to learn more about kernel modules, such as creating them and loading them directly onto the kernel of an OS. Developing kernel modules is a very good way to interact with the kernel by directly invoking kernel functions. The only slight downfall is that one must be very careful with kernel calls, because any error can crash the OS. Therefore, the assignment will be ran using a virtual machine to prevent the Host OS from crashing if there are any errors in the code. There are 2 parts to this project: The first consists of creating and inserting a kernel module into the Linux kernel, the second is to modify kernel modules so that they use linked-list array structures.

Part 1

For the first part, a model was created and inserted into the Linux Kernel. The module was written in the C language. The libraries init.h, module.h and kernel.h are required by the project to be able to start and execute the program into the kernel. The module_init() allows us to load the module on the kernel, while the module_exit() function is called when it is removed. These are the functions required for the first part. Once the C project is completed, the make command will be used to execute the makefile instructions that will execute the C project. This creates a project.ko file, which is the compiled kernel module that will be sent to the kernel. To load the module, the command "sudo insmod project.ko" is used. To check that the module has been properly loaded we used the command "lsmod" to list all the module that are currently inside the kernel. As we can see in the figure below, "project1" has been successfully loaded onto the kernel.



Furthermore, one can look at the kernel debug messages to see if the instructions of the C project have been executed using the "dmesg" command. The figure below shows how it should look.



Finally, to remove the module from the kernel we use the "sudo rmmod project1.ko" and to make sure it has been done we use the "lsmod" again to see that the module does not appear in the list anymore.

Part 2

In the second part of this project, the project module had to be modified so that the kernel makes use of a linked-list array structure. The linked-list structure is available to for kernel development in the <Linux/list.h> library. Therefore, at the beginning of the program this library needs to be called. A structure was used to create the containers for the elements in the array. The structure is as follows:

```
struct birthday {
```

```
        int day;
        int month;
        int year;
        struct list_head list;
};
```

The list_head structure is used to embed the linked list within the nodes that makes the list. Essentially, it has two elements which are “next” and “prev” which determine the next and previous elements in the list. To insert elements in the list, a list_head object needs to be created, which will be used as a reference to the head of the list. After that, instances of the struct “birthday” are created. To be able to create such structures, memory needs to be allocated for each object. The “kmalloc()” function is used for such purpose and it is equivalent to the malloc() function of high-level language but for kernel memory. To call this function, one must have included the <slab.h> library into the code.

The program then traverses the linked-list using the list_for_each_entry() macro that has the following input parameters: a pointer to the list structure, a pointer to the head and the name of the variable containing the list. The information is the printed using the “printk” function. To remove the elements from the list, the list_del() macro is used. This function removes an element from the list, while keeping the rest intact. To remove all elements, a loop can used to traverse the list and delete each element with that function. When removing an element, one must not forget to deallocate the memory it used in the kernel using the kFree() function from the slab.h library.

Making use of the first part program, we can see all this happening in the kernel using the dmesg command and displaying the debug messages. The first figure shows how the modules are loading when the program is loaded into the kernel, while the second figure displays what happens when removing the module.

```
[  432.831687] Loading Module
[  432.831691] Birthdays:
[  432.831693] day: 22, month: 1, year:2018
[  432.831695] day: 23, month: 1, year:2018
[  432.831697] day: 24, month: 1, year:2018
[  432.831698] day: 25, month: 1, year:2018
[  432.831700] day: 26, month: 1, year:2018
```

```
[  463.560508] Removing Module
[  463.560508] Removing: day: 22, month: 1, year:2018
[  463.560508] Removing: day: 23, month: 1, year:2018
[  463.560508] Removing: day: 24, month: 1, year:2018
[  463.560508] Removing: day: 25, month: 1, year:2018
[  463.560508] Removing: day: 26, month: 1, year:2018
oscreader@OSC:~/Desktop/osc9e-src/Assignment1$
```

Conclusion

In conclusion, this project has helped us better understand how to manipulate modules inside a kernel system, by making use of the module_init() and module_exit() commands. We have learned how to mount modules onto kernel and how to remove them by typing commands into the terminal. Finally, we have learned how to make the kernel module use data-structures such as the linked-list array and how memory allocation works inside the kernel.

Code

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/list.h>
#include <linux/types.h>
#include <linux/slab.h>

#define NUM_OF_PEOPLE 5

/*Birthday Structure*/
struct birthday {
	int day;
	int month;
	int year;
	struct list_head list;
};

/*Initializes and declares the head of the linked list*/
static LIST_HEAD(birthday_list);

/* This function is called when the module is loaded. */
int project1_init(void) {
	struct birthday *person, *ptr;
	int i;

	printk(KERN_INFO "Loading Module\n");

	/*Adds 5 birthdays to the kernel*/
	for(i=0; i < NUM_OF_PEOPLE; i++) {
		/*Requests malloc to the kernel*/
		person = kmalloc(sizeof(*person), GFP_KERNEL);
		/*gives the value to the birthday struct*/
		person->day = 22+i;
		person->month= 1;
		person->year = 2018;
		/*initializes the list*/
		INIT_LIST_HEAD(&person->list);

		/*Moves the struct to the end of the list (tail)*/
		list_add_tail(&person->list, &birthday_list);
	}

	/*goes through the list and prints all the birthdays*/
	printk(KERN_INFO "Birthdays:\n");

	list_for_each_entry(ptr, &birthday_list, list) {
		printk(KERN_INFO "day: %d, month: %d, year:%d\n", ptr->day, ptr->month,
ptr->year);
	}

	return 0;
}
```

```
/* This function is called when the module is removed. */
void project1_exit(void) {
	struct birthday *ptr,*next;

	/*goes through the list and removes all the birthdays and prints them*/
	printk(KERN_INFO "Removing Module\n");

	list_for_each_entry_safe(ptr,next,&birthday_list,list) {
		printk(KERN_INFO "Removing: day: %d, month: %d, year:%d\n", ptr->day,
ptr->month, ptr->year);
		list_del(&ptr->list);
		kfree(ptr);
	}
}

/* Macros for registering module entry and exit points. */
module_init(project1_init);
module_exit(project1_exit);

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Project1 Module");
MODULE_AUTHOR("SGG");
```