University of Hamburg
Department of Informatics
Group of Scientific Computing
K. Chasapis, M. Kuhn, M. Wiedemann

Praktikum „Kernel Programming“
in WiSe 2015/2016
Assignment 2
Deadline: 13.01.2016

## 1 Exercise: Debugging Techniques (25 pt.)

Debugging techniques in the Linux kernel differ from the user space. As we show in the lecture time there are several methods that can be used. For this exercise you will have to implement a *kprobe* module and also to make use of *debugfs*.

Your *kprobe* module should print the arguments of every *read, write and lseek* function calls of the kernel module that you implemented at the exercise 5 of the 1st assignment.

For the second part of this exercise, you will have to extend the exercise 5 of the 1st assignment. You are required is to export the same functionality offered by the *IOCTLs* using *debugfs*. In more detail you should create a per device *debugfs* file that will allow you to read and modify the size of the circular buffer.

## 2 Exercise: Threads Safety (20 pt.)

Many user space processes/threads can enter the Linux kernel simultaneously. Thus one has to take special care when programming kernel modules with multi-threaded support. Choose the proper synchronization method offered by the Linux kernel to solve the below tasks.

Extend the *kprobe* module that you have implemented in the previous exercise 1 in the following manner. First, add thread safe counters for every *read, write and lseek* function call. Second, allow counters reset functionality using an *IOCTL* command and a *debugfs* file.

## 3 Exercise: Memory Management (30 pt.)

The Linux kernel offers various memory allocation methods depending on the size and type of the allocation unit. The most commonly used are: *kmalloc, vmalloc, page_allocator* and *slab allocator.*

In this exercise you will have to implement a memory allocation wrapper that will call the appropriate allocation method depending on the requested size. The allocation function will only return the pointer to the available memory. You should keep track on the method that you used to allocate in order to call the appropriate function when you release the memory.

Allocation functions prototypes examples:

```
void * my_kernel_alloc( size_t size);

void * my_kernel_free( void * ptr);
```

Apart from that, add a *debugfs* file that will report the memory that has been allocated using this module (remember that the function calls can occur in parallel). Implement the above functionality inside a kernel module and offer the functionality to other kernel modules using the appropriate "EXPORT" macros.

Test the allocation wrapper by replacing the allocation calls of exercise 5 from the 1st assignment. Apart from the memory allocation module you should submit the modified version of the exercise 5 from the 1st assignment.

## 4 Exercise: Dynamic Kernel Module Support (25 pt.)

The Linux kernel modules can be loaded into the kernel only if they have be build against the same kernel version that it is running in the system. The Dynamic Kernel Module Support (DKMS) framework offers an automated method to recompile the kernel modules that reside outside of the kernel tree for every newly installed kernel. For this exercise, you have to add DKMS support to the exercise 2 of this assignment.

## Submission

You should submit the solution of above exercises in a single tar file named after your last name and the assignment number (e.g chasapis_2.tar). You should include main functions and also test programs. Submit the solution of your exercise by email to your corresponding supervisor. More details of this assignment will be given during the lecture time. You are encouraged to send questions to our mailing list.