CS342 – Operating Systems

Project 4

Section 2

Barış Ardıç 21401578

Emir Acımış 21201233

Part 1)

We skipped the first part, due to the poor performance of our virtual machine. (VirtualBox to be exact) Compiling a new kernel at that point was not time efficient considering it was optional. Our kernel version verified by “uname -r” **is 4.8.0-49-generic.**

Part 2)

We developed a simple hello module for the second part as the following :

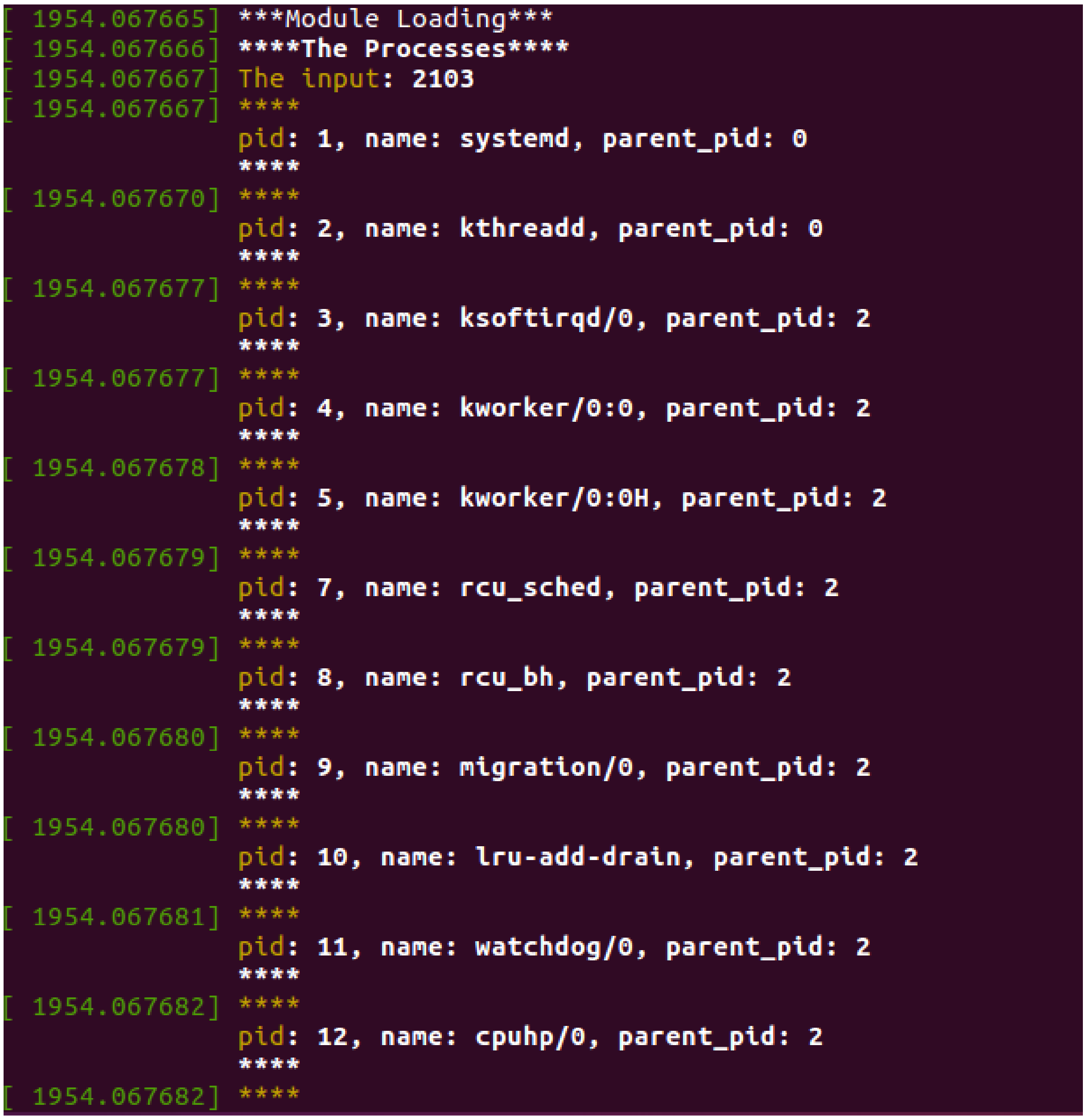
```
// hello.c - Simple kernel module
// barış ardıç-emir acımış
#include <linux/module.h>
#include <linux/kernel.h>


int init_module(void)// module starts from here when its loaded by
"insmod"
{
        printk(KERN_INFO "Hello world 1.\n");// printed string can be
found at /var/log/syslog

        return 0;// module is done.
}

void cleanup_module(void)// this is called when we remove the module
with rmmod.
{
        printk(KERN_INFO "Goodbye world 1.\n");
}
```

Part 3.1)

We are asked to print the process tree. We are printing the process tree by accessing the internals of task struct like following :

```
task->pid, task->comm, task->parent->pid
```

Then these properties are printed using "for each process" method defined in header file "sched.h" with the following as an example output. (Shown by "dmesg")



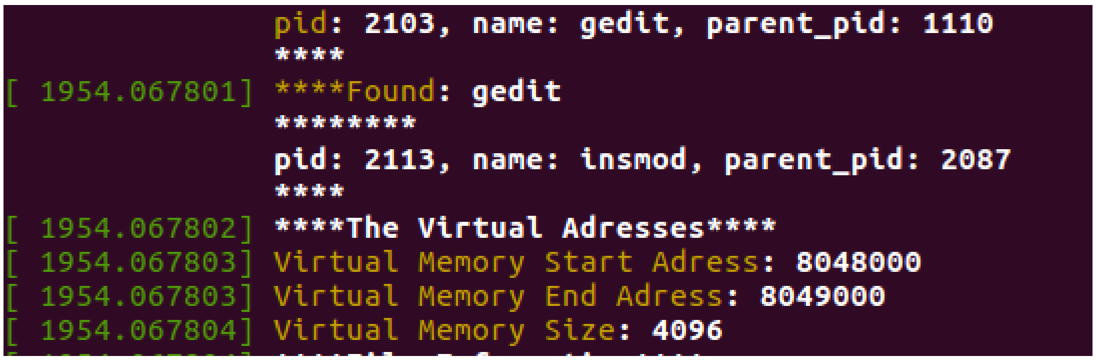
```
[ 1954.067665] ***Module Loading***
[ 1954.067666] ****The Processes****
[ 1954.067667] The input: 2103
[ 1954.067667] ****
               pid: 1, name: systemd, parent_pid: 0
               ****
[ 1954.067670] ****
               pid: 2, name: kthreadd, parent_pid: 0
               ****
[ 1954.067677] ****
               pid: 3, name: ksoftirqd/0, parent_pid: 2
               ****
[ 1954.067677] ****
               pid: 4, name: kworker/0:0, parent_pid: 2
               ****
[ 1954.067678] ****
               pid: 5, name: kworker/0:0H, parent_pid: 2
               ****
[ 1954.067679] ****
               pid: 7, name: rcu_sched, parent_pid: 2
               ****
[ 1954.067679] ****
               pid: 8, name: rcu_bh, parent_pid: 2
               ****
[ 1954.067680] ****
               pid: 9, name: migration/0, parent_pid: 2
               ****
[ 1954.067680] ****
               pid: 10, name: lru-add-drain, parent_pid: 2
               ****
[ 1954.067681] ****
               pid: 11, name: watchdog/0, parent_pid: 2
               ****
[ 1954.067682] ****
               pid: 12, name: cpuhp/0, parent_pid: 2
               ****
[ 1954.067682] ****
```

Part 3.2)

We are asked here the print the virtual memory layout information of a given process by a parameter to our kernel module. (Our variable for this is myPid) We match the parameter pid with the actual process pid then store it in a pointer of type “struct task”. If there is no process with given pid we print nothing here. The access to the VM layout is done like the following :

```
match->mm->mmap->vm_start);
match->mm->mmap->vm_end);
match->mm->mmap->vm_end - match->mm->mmap->vm_start));
```

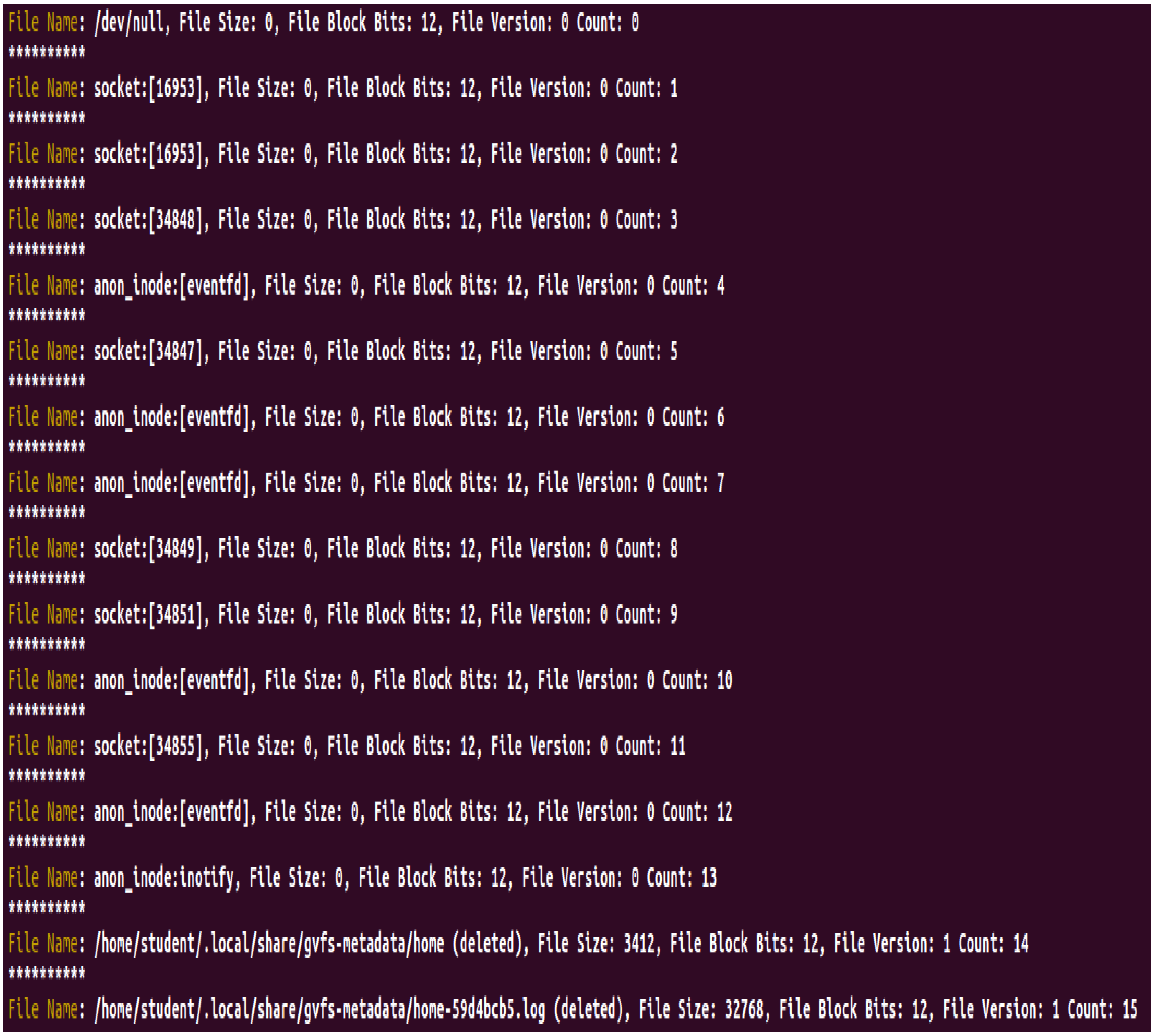
Here match is the process with the given pid and sample output is the following :



```
                  pid: 2103, name: gedit, parent_pid: 1110
                  ****
[ 1954.067801] ****Found: gedit
                  ********
                  pid: 2113, name: insmod, parent_pid: 2087
                  ****
[ 1954.067802] ****The Virtual Adresses****
[ 1954.067803] Virtual Memory Start Adress: 8048000
[ 1954.067803] Virtual Memory End Adress: 8049000
[ 1954.067804] Virtual Memory Size: 4096
```

Part 3.2)

We are asked here to print the information of the open files of the given process from its PCB. We accessed the file name, file size, block bits, file version and where the file lies in the linked list.



```
File Name: /dev/null, File Size: 0, File Block Bits: 12, File Version: 0 Count: 0
**********
File Name: socket:[16953], File Size: 0, File Block Bits: 12, File Version: 0 Count: 1
**********
File Name: socket:[16953], File Size: 0, File Block Bits: 12, File Version: 0 Count: 2
**********
File Name: socket:[34848], File Size: 0, File Block Bits: 12, File Version: 0 Count: 3
**********
File Name: anon_inode:[eventfd], File Size: 0, File Block Bits: 12, File Version: 0 Count: 4
**********
File Name: socket:[34847], File Size: 0, File Block Bits: 12, File Version: 0 Count: 5
**********
File Name: anon_inode:[eventfd], File Size: 0, File Block Bits: 12, File Version: 0 Count: 6
**********
File Name: anon_inode:[eventfd], File Size: 0, File Block Bits: 12, File Version: 0 Count: 7
**********
File Name: socket:[34849], File Size: 0, File Block Bits: 12, File Version: 0 Count: 8
**********
File Name: socket:[34851], File Size: 0, File Block Bits: 12, File Version: 0 Count: 9
**********
File Name: anon_inode:[eventfd], File Size: 0, File Block Bits: 12, File Version: 0 Count: 10
**********
File Name: socket:[34855], File Size: 0, File Block Bits: 12, File Version: 0 Count: 11
**********
File Name: anon_inode:[eventfd], File Size: 0, File Block Bits: 12, File Version: 0 Count: 12
**********
File Name: anon_inode:inotify, File Size: 0, File Block Bits: 12, File Version: 0 Count: 13
**********
File Name: /home/student/.local/share/gvfs-metadata/home (deleted), File Size: 3412, File Block Bits: 12, File Version: 1 Count: 14
**********
File Name: /home/student/.local/share/gvfs-metadata/home-59d4bcb5.log (deleted), File Size: 32768, File Block Bits: 12, File Version: 1 Count: 15
```

These properties are accessed like the following:

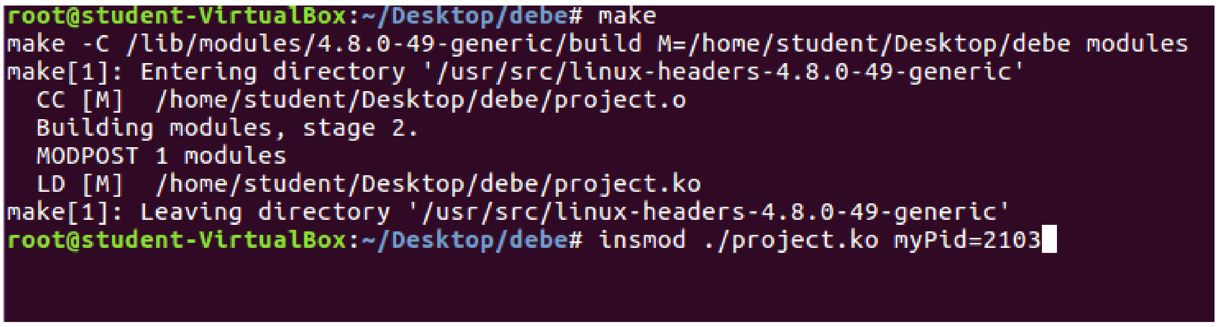
```
path = &(match -> files ->fd_array[count]->f_path);
name = d_path(path, buf, sizeof(buf));
```

Buffer here is a char array and its size is “100”. Name is returned as a string by d_path. The acces for the rest of the information is more straightforward:

```
match->files->fd_array[count]->f_inode->i_size,
match->files->fd_array[count]->f_inode->i_blkbits,
match->files->fd_array[count]->f_inode->i_version,
count);
```

Count here is our loop index and it iterates from 0 to number of open files that the process has when the module is loaded. That variable is accessed from the file_table.

- Source code of our module is appended in the next page followed by the makefile.
- We tested the correctness of the output with command line commands and kernel logs such as “ps aux” , “lsof -p “pid” ”, “cat /proc/modules”,”dmesg”.
- For the compilation “sudo make” did not seem to work after “sudo bash” “make” works fine with gcc compiler.
- Here is an example run :



```
root@student-VirtualBox:~/Desktop/debe# make
make -C /lib/modules/4.8.0-49-generic/build M=/home/student/Desktop/debe modules
make[1]: Entering directory '/usr/src/linux-headers-4.8.0-49-generic'
  CC [M]  /home/student/Desktop/debe/project.o
  Building modules, stage 2.
  MODPOST 1 modules
  LD [M]  /home/student/Desktop/debe/project.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-49-generic'
root@student-VirtualBox:~/Desktop/debe# insmod ./project.ko myPid=2103
```

```
#include <linux/sched.h>
#include <linux/init.h>
#include <linux/fdtable.h>
#include <linux/fs.h>
#include <linux/fs_struct.h>
#include <linux/dcache.h>
#include <linux/slab.h>
#include <linux/mm.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Emir Acimis && Baris Ardic");

static int myPid = 1;
module_param(myPid, int, 0000);
void project_function(struct task_struct *task)
{
        struct task_struct *child;
        struct task_struct *match;
        struct list_head *list;
        struct fdtable *files_table;
        int count = 0;
        char buf[100];
        struct path *path;
        char* name = NULL;


        //PART3.1
        printk("****The Processes****\n");
        printk("The input: %d\n", myPid);
        for_each_process(task){
                printk("****\npid: %d, name: %s, parent_pid: %d\n****\n", task->pid, task->comm, task-
>parent->pid);

                if(task->pid == myPid){
                        printk("****Found: %s\n****", task->comm);
                        match = task;
                }

                list_for_each(list, &task->children) {
                        child = list_entry(list, struct task_struct, sibling);
                }
        }//for each end
        printk("****The Virtual Adresses****\n");
        //PART3.2
        if(match->mm->mmap){
                printk("Virtual Memory Start Adress: %lx\n",match->mm->mmap->vm_start);
                printk("Virtual Memory End Adress: %lx\n",match->mm->mmap->vm_end);
                printk("Virtual Memory Size: %lu\n",(match->mm->mmap->vm_end - match->mm-
>mmap->vm_start));
        }
```

```
	//PART3.3
	printk("****File Information****\n");
	if(match->files != NULL){
		files_table = files_fdtable(match->files);
		if(files_table != NULL && files_table -> fd != NULL){
			printk("Open files number: %d\n", files_table -> max_fds);
			while(count < files_table -> max_fds){
				if(match->files->fd_array[count]   !=   NULL   &&   match->files-
>fd_array[count]->f_inode != NULL){
					path = &(match -> files ->fd_array[count]->f_path);
					name = d_path(path, buf, sizeof(buf));
					printk("File Name: %s, File Size: %lld, File Block Bits: %hu, File
Version:  %llu  Count:  %d\n",name  ,  match->files->fd_array[count]->f_inode->i_size,  match->files-
>fd_array[count]->f_inode->i_blkbits ,match->files->fd_array[count]->f_inode->i_version, count);
					printk("**********\n");
				}
				count = count + 1;
			}
		}
	}
}

int task_lister_init(void)
{
	printk(KERN_INFO "***Module Loading***\n");
	project_function(&init_task);
	return 0;
}

void task_lister_exit(void)
{
	printk(KERN_INFO "***Module Removing***\n");
}

module_init(task_lister_init);
module_exit(task_lister_exit);


// makefile here.

obj-m += project.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
```