CS353: Linux Kernel Project 3 Report

Weichen Li
5120309662
eizo.lee@sjtu.edu.cn
Department of Computer
Science and Engineering
School of Electronic Information
and Electrical Engineering
Shanghai Jiao Tong University

CONTENTS

I. OBJECT

- Write a module that is called *mtest*.
- When module loaded, module will create a proc fs entry */proc/mtest*.
- */proc/mtest* will accept 3 kinds of input:
  - *"listvma"* will print all vma of current process in the format of

    ```
    start-addr   end-addr   permission
    0x10000      0x20000    rwx
    0x30000      0x40000    r–
    ...          ...        ...
    ```
  - *"findpage addr"* will find $va \rightarrow pa$ translation of address in current process's mm context and print it. If there is not $va \rightarrow pa$ translation, printk "translation not found".
  - *"writeval addr val"* will change an unsigned long size content in current process's virtual address into val. Note module should write to identity mapping address of addr and verify it from userspace address addr.
  - Implement a program, in which there is an integer variable *v*. The initial value of *v* is 0. The program should recognize commands such as *"write int"* and *"print"*. The former can write to the file *"/proc/mtest"*, the latter can output the value of *v*.
- All the print can be done with printk and check result with *dmesg*.

II. COMPILE STEPS

A. Preprocess working

At the very beginning, to save some trivial operations on authority verification, we need get the root authority first. (Need to input the key)

```
su
```

B. Create relevant file

To start our work, we need to create a new directory to store our work files.

```
mkdir /usr/src/Project3
cd /usr/src/Project3
```

1) mtest.c: Then we create a *.c* file.

```
vi mtest.c
```

Every time the “vi” command is called, the terminal will jump into the command window of VIM. We can simply input “:wq” and press “Enter” button to get back to the terminal.

Next we need to create a Makefile file:

```
vi Makefile
```

Do not quit the VIM in a hurry, we need to add some content in the Makefile file, press the “i” button on the keyboard first, then input:

```
obj-m := mtest.o

KDIR := /lib/modules/$(shell uname -r)/build

PWD := $(shell pwd)

all:

make -C $(KDIR) M=$(PWD) modules

clean:

rm *.o *.ko *.mod.c Module.symvers modules.order -f
```

Then press the “ESC” button, input “:wq”, press “Enter” button, the Makefile file is built. (This is the typical way to modify and save text files using VIM, “wq” means save and quit)

Next, input:

```
vi mtest.c
```

Add content below , save and quit.

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <linux/string.h>
#include <linux/vmalloc.h>
#include <linux/sched.h>
#include <linux/init.h>
#include <linux/slab.h>
#include <linux/mm.h>
#include <linux/vmalloc.h>
#include <linux/highmem.h>
#include <asm/uaccess.h>
#include <linux/errno.h>
#include <linux/fs.h>

static void mtest_dump_vma_list(void)

{

struct task_struct *task = current;                //get the task_struct of the current process

struct mm_struct *mm = task->mm;

struct vm_area_struct *vma;                //get the vma area of the current process

int count = 0;     //the number of vma

down_read(&mm->mmap_sem);
```

```
for(vma = mm->mmap; vma; vma = vma->vm_next)
{
count++;

printk("%d:  0x%lx 0x%lx ", count, vma->vm_start, vma->vm_end);

if (vma->vm_flags & VM_READ)
printk("r");
else
printk("-");

if (vma->vm_flags & VM_WRITE)
printk("w");
else
printk("-");

if (vma->vm_flags & VM_WRITE)
printk("x");
else
printk("-");

printk("\n");
}

up_read(&mm->mmap_sem);

}
static struct page *
my_follow_page(struct vm_area_struct *vma, unsigned long addr)
{
pgd_t *pgd;
pmd_t *pmd;
pud_t *pud;
pte_t *pte;

spinlock_t *ptl;
```

```
struct page *page = NULL;

struct mm_struct *mm = vma->vm_mm;



pgd = pgd_offset(mm, addr);     //get pgd

if (pgd_none(*pgd) || unlikely(pgd_bad(*pgd)))

goto out;


pud = pud_offset(pgd, addr);   //get pud

if (pud_none(*pud) || unlikely(pud_bad(*pud)))

goto out;


pmd = pmd_offset(pud, addr);   //get pmd

if (pmd_none(*pmd) || unlikely(pmd_bad(*pmd)))

goto out;


pte = pte_offset_map_lock(mm, pmd, addr, &ptl); //get pte


if (!pte)

goto out;


if (!pte_present(*pte))   //pte not in memory

goto unlock;


page = pfn_to_page(pte_pfn(*pte));


if (!page)

goto unlock;

get_page(page);


unlock:

pte_unmap_unlock(pte, ptl);


out:

return page;


}
```

```
static void mtest_find_page(unsigned long addr)
{
struct vm_area_struct *vma;
struct task_struct *task = current;
struct mm_struct *mm = task->mm;
unsigned long kernel_addr;
struct page *page;

down_read(&mm->mmap_sem);
vma = find_vma(mm, addr);
page = my_follow_page(vma, addr);

if (!page)
{
printk("translation failed.\n");
goto out;
}

kernel_addr = (unsigned long) page_address(page);

kernel_addr += (addr & ~PAGE_MASK);

printk("vma 0x%lx -> pma 0x%lx\n", addr, kernel_addr);

out:
up_read(&mm->mmap_sem);
}
static void
mtest_write_val(unsigned long addr, unsigned long val)
{
struct vm_area_struct *vma;
struct task_struct *task = current;
struct mm_struct *mm = task->mm;
struct page *page;
unsigned long kernel_addr;
```

```
down_read(&mm->mmap_sem);
vma = find_vma(mm, addr);

//test if it is a legal vma
if (vma && addr >= vma->vm_start && (addr + sizeof(val)) < vma->vm_end)
{
if (!(vma->vm_flags & VM_WRITE))   //test if we have rights to write
{
printk("cannot write to 0x%lx\n", addr);
goto out;
}

page = my_follow_page(vma, addr);
if (!page)
{
printk("page not found 0x%lx\n", addr);
goto out;
}

kernel_addr = (unsigned long) page_address(page);
kernel_addr += (addr &~ PAGE_MASK);
printk("write 0x%lx to address 0x%lx\n", val, kernel_addr);
*(unsigned long *)kernel_addr = val;
put_page(page);
}
else
{
printk("no vma found for %lx\n", addr);
}
out:
up_read(&mm->mmap_sem);
}
static ssize_t
mtest_write(struct file *file, const char __user *buffer, size_t count, loff_t *data)
{
char buf[128];
unsigned long val, val2;
```

```
if (count > sizeof(buf))

return -EINVAL;


if (copy_from_user(buf, buffer, count))    //get the command from shell

return -EINVAL;


if (memcmp(buf, "listvma", 7) == 0)

mtest_dump_vma_list();

else if (memcmp(buf, "findpage", 8) == 0)

{

if (sscanf(buf+8, "%lx", &val) == 1)

mtest_find_page(val);

}

else if (memcmp(buf, "writeval", 8) == 0)

{

if (sscanf(buf+8, "%lx %lx", &val, &val2) == 2)

{

mtest_write_val(val, val2);

}

}

return count;

}


static struct

file_operations proc_mtest_operation = {

write: mtest_write,

};


static int __init

mtest_init(void)

{

proc_create("mtest", 0, NULL, &proc_mtest_operation);

printk("Create mtest...\n");

return 0;

}
```

```
        static void __exit

        mtest_exit(void)

        {

        remove_proc_entry("mtest", NULL);

        }


        MODULE_LICENSE("GPL");

        MODULE_DESCRIPTION("memory management task");

        module_init(mtest_init);

        module_exit(mtest_exit);
```

2) program.c: Then we create a *.c* file.

```
vi program.c
```

Every time the "vi" command is called, the terminal will jump into the command window of VIM. We can simply input ":wq" and press "Enter" button to get back to the terminal. To edit the file, we can press "i".

Add content below , save and quit.

```
#include<stdio.h>

int main()
{
        int n = 0;
        int i = 0;
        int v = 0;
        char s[80];
        FILE *tt;
        printf("Virtual Memory address of v: %p\n", &v);

        while(1){
                tt = fopen("/proc/mtest", "w+");
                fgets(s, 80, stdin);
                if(memcmp(s, "exit", 4) == 0) break;
                else if(memcmp(s, "print", 5) == 0)
                        printf("%d\n", v);
                else if(memcmp(s, "write", 5) == 0){
                        sscanf(s, "write %d", &i);
                        n = fprintf(tt, "writeval%lx %lx", (unsigned long)(void*)&v, (unsigned long)i);
                }
                fclose(tt);
        }
        return 0;
}
```

Then press the "ESC" button, input ":wq", press "Enter" button, the Makefile file is built. (This is the typical way to modify and save text files using VIM, "wq" means save and quit)

C. Compile and insert the module

So far we have completed all the preparation. Next, input:

```
        make
```

Then input:

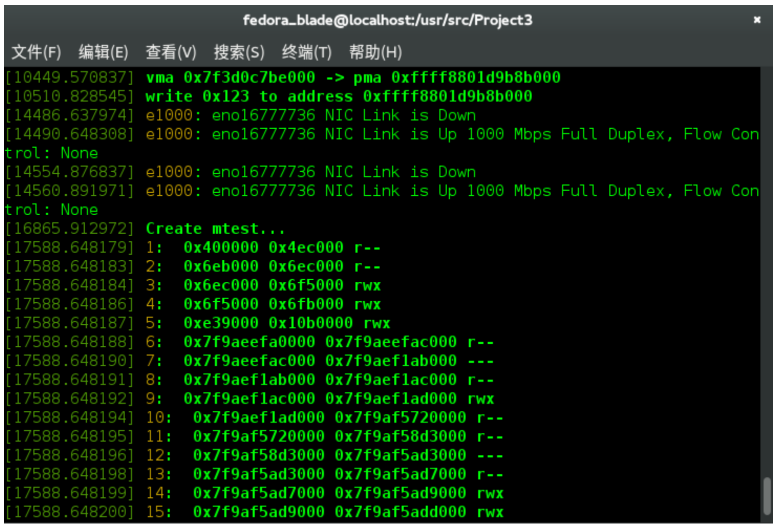
```
        insmod mtest.ko
```

D. *Test the module and program*

1) *listvma:* Input

```
        echo "listvma" > /proc/mtest
        dmesg
```

Then we will get the output shown as Fig 1.


fedora_blade@localhost:/usr/src/Project3
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

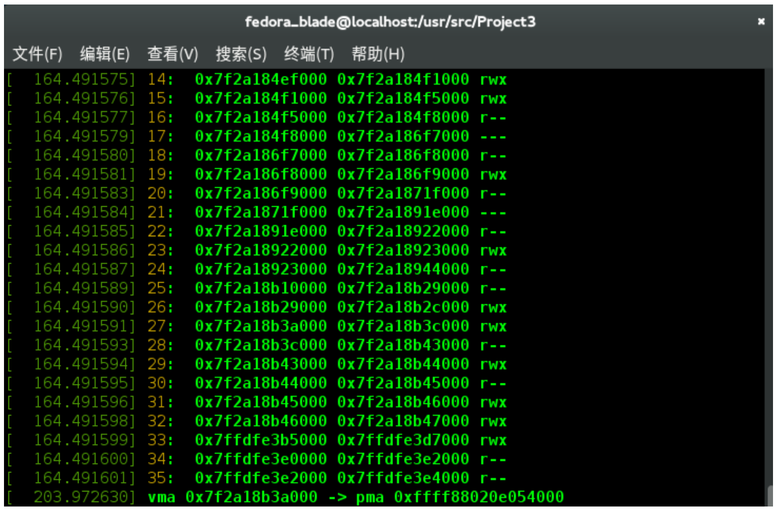
```
[10449.570837] vma 0x7f3d0c7be000 -> pma 0xffff8801d9b8b000
[10510.828545] write 0x123 to address 0xffff8801d9b8b000
[14486.637974] e1000: eno16777736 NIC Link is Down
[14490.648308] e1000: eno16777736 NIC Link is Up 1000 Mbps Full Duplex, Flow Con
trol: None
[14554.876837] e1000: eno16777736 NIC Link is Down
[14560.891971] e1000: eno16777736 NIC Link is Up 1000 Mbps Full Duplex, Flow Con
trol: None
[16865.912972] Create mtest...
[17588.648179] 1:  0x400000 0x4ec000 r--
[17588.648183] 2:  0x6eb000 0x6ec000 r--
[17588.648184] 3:  0x6ec000 0x6f5000 rwx
[17588.648186] 4:  0x6f5000 0x6fb000 rwx
[17588.648187] 5:  0xe39000 0x10b0000 rwx
[17588.648188] 6:  0x7f9aeefa0000 0x7f9aeefac000 r--
[17588.648190] 7:  0x7f9aeefac000 0x7f9aef1ab000 ---
[17588.648191] 8:  0x7f9aef1ab000 0x7f9aef1ac000 r--
[17588.648192] 9:  0x7f9aef1ac000 0x7f9aef1ad000 rwx
[17588.648194] 10:  0x7f9aef1ad000 0x7f9af5720000 r--
[17588.648195] 11:  0x7f9af5720000 0x7f9af58d3000 r--
[17588.648196] 12:  0x7f9af58d3000 0x7f9af5ad3000 ---
[17588.648198] 13:  0x7f9af5ad3000 0x7f9af5ad7000 r--
[17588.648199] 14:  0x7f9af5ad7000 0x7f9af5ad9000 rwx
[17588.648200] 15:  0x7f9af5ad9000 0x7f9af5add000 rwx
```

Fig. 1. listvma

2) *findpage:* Input

```
        echo "findpage 0x..." > /proc/mtest
        dmesg
```

Then we will get the output shown as Fig 2.


fedora_blade@localhost:/usr/src/Project3
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

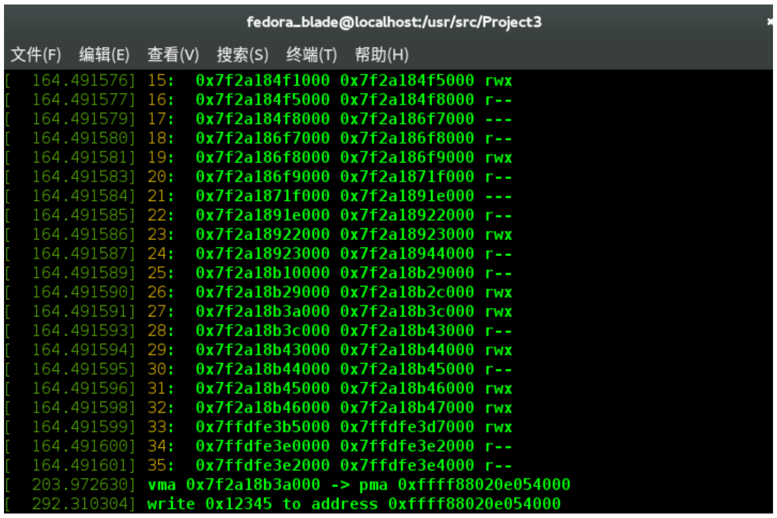
```
[  164.491575] 14:  0x7f2a184ef000 0x7f2a184f1000 rwx
[  164.491576] 15:  0x7f2a184f1000 0x7f2a184f5000 rwx
[  164.491577] 16:  0x7f2a184f5000 0x7f2a184f8000 r--
[  164.491579] 17:  0x7f2a184f8000 0x7f2a186f7000 ---
[  164.491580] 18:  0x7f2a186f7000 0x7f2a186f8000 r--
[  164.491581] 19:  0x7f2a186f8000 0x7f2a186f9000 rwx
[  164.491583] 20:  0x7f2a186f9000 0x7f2a1871f000 r--
[  164.491584] 21:  0x7f2a1871f000 0x7f2a1891e000 ---
[  164.491585] 22:  0x7f2a1891e000 0x7f2a18922000 r--
[  164.491586] 23:  0x7f2a18922000 0x7f2a18923000 rwx
[  164.491587] 24:  0x7f2a18923000 0x7f2a18944000 r--
[  164.491589] 25:  0x7f2a18b10000 0x7f2a18b29000 r--
[  164.491590] 26:  0x7f2a18b29000 0x7f2a18b2c000 rwx
[  164.491591] 27:  0x7f2a18b3a000 0x7f2a18b3c000 rwx
[  164.491593] 28:  0x7f2a18b3c000 0x7f2a18b43000 r--
[  164.491594] 29:  0x7f2a18b43000 0x7f2a18b44000 rwx
[  164.491595] 30:  0x7f2a18b44000 0x7f2a18b45000 r--
[  164.491596] 31:  0x7f2a18b45000 0x7f2a18b46000 rwx
[  164.491598] 32:  0x7f2a18b46000 0x7f2a18b47000 rwx
[  164.491599] 33:  0x7ffdfe3b5000 0x7ffdfe3d7000 rwx
[  164.491600] 34:  0x7ffdfe3e0000 0x7ffdfe3e2000 r--
[  164.491601] 35:  0x7ffdfe3e2000 0x7ffdfe3e4000 r--
[  203.972630] vma 0x7f2a18b3a000 -> pma 0xffff88020e054000
```

Fig. 2. findpage

3) *writeval:* Input

```
echo "writeval 0x... ..." > /proc/mtest
dmesg
```

Then we will get the output shown as Fig 3.
At this moment, we have successfully completed all the tasks for experiment 3, congratulations!


fedora_blade@localhost:/usr/src/Project3
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
[  164.491576] 15:  0x7f2a184f1000 0x7f2a184f5000 rwx
[  164.491577] 16:  0x7f2a184f5000 0x7f2a184f8000 r--
[  164.491579] 17:  0x7f2a184f8000 0x7f2a186f7000 ---
[  164.491580] 18:  0x7f2a186f7000 0x7f2a186f8000 r--
[  164.491581] 19:  0x7f2a186f8000 0x7f2a186f9000 rwx
[  164.491583] 20:  0x7f2a186f9000 0x7f2a1871f000 r--
[  164.491584] 21:  0x7f2a1871f000 0x7f2a1891e000 ---
[  164.491585] 22:  0x7f2a1891e000 0x7f2a18922000 r--
[  164.491586] 23:  0x7f2a18922000 0x7f2a18923000 rwx
[  164.491587] 24:  0x7f2a18923000 0x7f2a18944000 r--
[  164.491589] 25:  0x7f2a18b10000 0x7f2a18b29000 r--
[  164.491590] 26:  0x7f2a18b29000 0x7f2a18b2c000 rwx
[  164.491591] 27:  0x7f2a18b3a000 0x7f2a18b3c000 rwx
[  164.491593] 28:  0x7f2a18b3c000 0x7f2a18b43000 r--
[  164.491594] 29:  0x7f2a18b43000 0x7f2a18b44000 rwx
[  164.491595] 30:  0x7f2a18b44000 0x7f2a18b45000 r--
[  164.491596] 31:  0x7f2a18b45000 0x7f2a18b46000 rwx
[  164.491598] 32:  0x7f2a18b46000 0x7f2a18b47000 rwx
[  164.491599] 33:  0x7ffdfe3b5000 0x7ffdfe3d7000 rwx
[  164.491600] 34:  0x7ffdfe3e0000 0x7ffdfe3e2000 r--
[  164.491601] 35:  0x7ffdfe3e2000 0x7ffdfe3e4000 r--
[  203.972630] vma 0x7f2a18b3a000 -> pma 0xffff88020e054000
[  292.310304] write 0x12345 to address 0xffff88020e054000
```

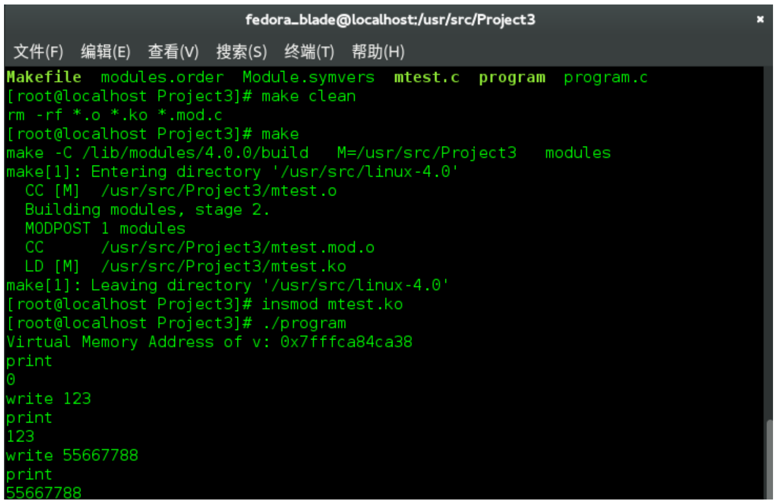
Fig. 3.   writeval

Tips: you can use “rmmod xxx.ko” command to remove module xxx from the kernel, or use “make clean” to delete all the files produced when compiling, then only the initial files left.

4) Program Execution: We can execute the program using codes below:

```
        ./program
```

then we can test the program as shown in Fig 4.


fedora_blade@localhost:/usr/src/Project3
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
Makefile  modules.order  Module.symvers  mtest.c  program  program.c
[root@localhost Project3]# make clean
rm -rf *.o *.ko *.mod.c
[root@localhost Project3]# make
make -C /lib/modules/4.0.0/build   M=/usr/src/Project3   modules
make[1]: Entering directory '/usr/src/linux-4.0'
  CC [M]  /usr/src/Project3/mtest.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /usr/src/Project3/mtest.mod.o
  LD [M]  /usr/src/Project3/mtest.ko
make[1]: Leaving directory '/usr/src/linux-4.0'
[root@localhost Project3]# insmod mtest.ko
[root@localhost Project3]# ./program
Virtual Memory Address of v: 0x7fffca84ca38
print
0
write 123
print
123
write 55667788
print
55667788
```

Fig. 4.   Program Execution