CS353 Linux Kernel Project Report

Bingyu Shen
5130309283
sby2013@sjtu.edu.cn

June 3, 2016

Contents

Abstract

In the projects of this lecture, I finished four parts of the project. The first project requires to upgrade the linux kernel to the newest version; the second project part A focus on writing module program and install it in the kernel program, part B focus on process management. which requires us to add a member in the $task\_struct$ and record how many times it has been executed. The third project focus on memory management, which requires to write a module $mtest$ and can write and find certain memory address of all running process. The fourth project focus on file system, in which we can learn a lot about the romfs file system. All programs test runs on Unbuntu 15.10 and results are checked by TA. The source and modified program are attached.

1 Compile the Linux Kernel

1.1 Requirements

Upgrade your linux kernel to newest version.

1.2 Experiment

I installed Ubuntu 15.10 in vmware workstation, in case the operating system is broken. The version of linux kernel on Ubuntu 15.10 is 4.2.0. I download the newest version of linux kernel on $www.kernel.org$, which is 4.4.3. As you know, it is quite difficult to upgrade the linux kernel from 3.x to 4.x, so I choose the newest version of Ubuntu.

A. Preparation
First download and extract kernel to $/usr/src$, so at this moment the folder looks like. Go to the kernel folder. If you are not the first time to compile, you may do the following

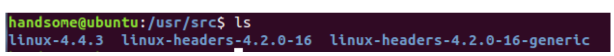


Figure 1: Environment preparation

command.

```
# make clean
# make mrproper
```

B. Compile
Then before you do make command, you should do make menu command. I strongly suggest you do the make menuconfig command.

```
# make manuconfig
```

Then you can customize your kernel. Since I just want the version to upgrade, I just use the default setting.
Next step is to compile and install the kernel.

```
# make
# make modules_install
# make install
```

The *make* step will take a long time, since I use the virtual machine, it usually takes 4-5 hours.

C. Update GRUB
Use the following command.

```
# cd /boot
# mkinitrd -o initrd.img-4.4.3
```

At this step, it fails to change the grub. I looked it up on the internet, and use the following command instead.

```
# sudo update-grub
```

It works!

1.3 Results

The kernel version is successfully changed.

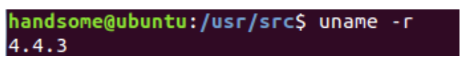


Figure 2: Kernel version

2 Module Program and Process Management

2.1 Part A: Module Program

2.1.1 Requirements

Compile a kernel and run it in the system.

A. Module 1: Load/unload the module can output some info

B. Module 2: Module accepts a parameter (an integer) and load the module, output the parameters value

C. Module 3: Module creates a proc file, reading the proc file returns some info

2.1.2 Experiment

A. Module 1
Load and unload the module, it can print some info. We know that to write a module, you need to create the init and exit function. In the init and exit function, you can use *printk* to print the kernel message. So the first module of program Module_1.c is as follows:

```c
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

static int __init hello_init(void){
        printk("<3>Hello a linux kernel module 2A_1.\n");
        return 0;
}

static void __exit hello_exit(void){
        printk("<3>Bye 2A_1.\n");
}

module_init(hello_init);
module_exit(hello_exit);
MODULE_LICENSE("GPL");
```

B. Module 2

The module can accept parameter when insmod. This requires you to use `module_param_array`.

```c
module_param_array(name, type, num, perm)
```

name is the parameter's name, it is usually a static variable in the module. The parameter can be set after install the module.
type is the type of parameter, it can be int, short, bool, etc.
num is a $int*$ type variable, and it stores the num of variables.
perm is the authority level of module. It is defined in $stat.h$.

Based on this information, we can write the second module, Module_2.c.

```c
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/moduleparam.h>
static int test[3];
int num;
module_param_array(test,int,&num,0644);
void hello_foo(void){
        printk("Hello\n");
}
EXPORT_SYMBOL(hello_foo);
static int __init hello_init(void){
        printk(KERN_INFO"Hello world\n");
        printk(KERN_INFO"Params:test:%d,%d,%d;\n",test[0],
           test[1],test[2]);
        return 0;
}
static void __exit hello_exit(void){
        printk(KERN_INFO"Goodbye world\n");
}
```

```
20 MODULE_LICENSE("GPL");
21 MODULE_DESCRIPTION("Test");
22 module_init(hello_init);
23 module_exit(hello_exit);
```

Notice that we defined a `static int test[3]` in the 5th line. It can be used to accept parameter when we install the module.
The framework of the program is just like the Module1. Just modify the *printk* to print the parameters in `test`.

C. Module 3
This module is much more harder than the previous ones.

```
#include <linux/module.h>
#include <linux/proc_fs.h>
#include <linux/seq_file.h>

static int hello_proc_show(struct seq_file *m, void *v) {
        seq_printf(m, "I am Bingyu Shen!\n");
        return 0;
}
static int hello_proc_open(struct inode *inode, struct file *file) {
        return single_open(file, hello_proc_show, NULL);
}
static const struct file_operations hello_proc_fops = {
        .owner = THIS_MODULE,
        .open = hello_proc_open,
        .read = seq_read,
        .llseek = seq_lseek,
        .release = single_release,
};
static int __init hello_proc_init(void) {
        proc_create("hello_proc", 0, NULL, &hello_proc_fops);
        return 0;
}
static void __exit hello_proc_exit(void) {
        remove_proc_entry("hello_proc", NULL);
}
MODULE_LICENSE("GPL");
module_init(hello_proc_init);
module_exit(hello_proc_exit);
```

This part requires the module to create a proc file, reading the proc file returns some info. It uses the function `proc_create` to create a proc file.

```
static inline struct proc_dir_entry *proc_create(const char *name, mode_t mode, \\
 struct proc_dir_entry *parent, const struct file_operations *proc_fops);
```

name is the proc name, mode_t pis the type of umask, it is unsigned octal number. It has three bits , each bit stands for the permission of a user. First bit for owner, second bit for group, third and last bit for others.

- Octal value : Permission
- 0 : read, write and execute
- 1 : read and write
- 2 : read and execute
- 3 : read only
- 4 : write and execute
- 5 : write only
- 6 : execute only
- 7 : no permissions

Now, you can use above table to calculate file permission. For example, if umask is set to 077, the permission can be calculated as follows:

- owner: read, write and execute
- group: No permissions
- other: No permissions

struct proc_dir_entry is a proc directory. It can be created use the following function

```
struct proc_dir_entry *proc_mkdir(const char *name, struct
    proc_dir_entry *parent);
```

If you want the parent directory is */proc*, just set parent = NULL.

file_operations is defined like

```
static const struct file_operations hello_proc_fops = {
        .owner = THIS_MODULE,
        .open = hello_proc_open,
        .read = seq_read,
        .llseek = seq_lseek,
        .release = single_release,
};
```

Note that parameter begin with seq_ and single_ are defined in kernel.
.open operation is a function to read a specific file.
.write operation is a function to write the proc file.

In the module 3, I just use 0 as the mode so that all users can do any operation on this proc. And directory is set to NULL so that is created in /proc directory. And I define the file_operation only with open operations. And it can print some info when open the file.

The makefile is written as follows.

```
obj-m := module_1.o module_2.o module_3.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
```

2.1.3 Results

The commands used are as follows.

```
# insmod module_2A_1.ko
# dmesg

# insmod module_2A_2.ko test=1,2,3
# dmesg

# insmod module_2A_3.ko
# cat /proc/hello_proc
```

And the results are listed below.

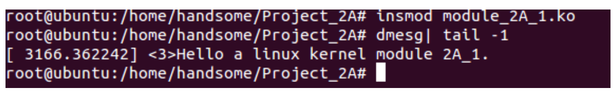


Figure 3: Module 1 output

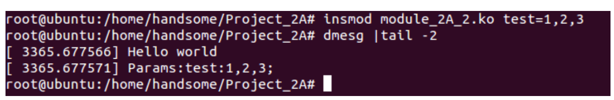


Figure 4: Module 2 output

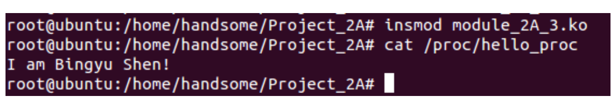


Figure 5: Module 3 output

2.2 Part B: Process

2.2.1 Requirements

Process: schedule in times.

A. Add ctx, a new member to *task_struct* to record the schedule in times of the process; When a task is scheduled in to run on a cpu, increase ctx of the process;

B. Export ctx under /proc/XXX/ctx;

2.2.2 Experiment

- First modify the kernel.

  A. add `ctx` to `task_struct`.
  `task_struct` is in `/include/linux/sched.h`
  The program after modification is like Figure. 6.

```
struct task_struct {
        /* bingyu shen */
        unsigned int ctx;
        /* bingyu shen*/
        volatile long state;    /* -1 unrunnable, 0 runnable, >0 stopped */
        void *stack;
        atomic_t usage;
        unsigned int flags;     /* per process flags, defined below */
        unsigned int ptrace;

#ifdef CONFIG_SMP
        struct llist_node wake_entry;
        int on_cpu;
        unsigned int wakee_flips;
        unsigned long wakee_flip_decay_ts;
        struct task_struct *last_wakee;

        int wake_cpu;
#endif
        int on_rq;

        int prio, static_prio, normal_prio;
        unsigned int rt_priority;
        const struct sched_class *sched_class;
        struct sched_entity se;
        struct sched_rt_entity rt;
#ifdef CONFIG_CGROUP_SCHED
        struct task_group *sched_task_group;
#endif
                                                        1381,19-32    43%
```

Figure 6: Add ctx to task_struct

  B. Initialize `ctx=0` in the `do_fork` function.
  The function of `do_fork is` in `kernel/fork.c.` After modification it is like Figure.7

  C. In kernel file `kernel/sched/core.c`, find the where processes are switched, plus `ctx` by 1.
  The program after modification is like Figure.8

  D. Create a new entry in the `pid_entry_tgid_base_stuff[]`. To specify a function to be executed.

```
ONE("ctx", S_IRUSR, proc_pid_my_file),
```

  Modify it like Figure. 9. And the function `proc_pid_my_file` is defined in the later, like Figure. 10

- CD to the directory of the linux kernel.(`/usr/src/linux-4.4.3`).Then compile the kernel with the following command.

```
# make
# make modules_install
# make install
```

The time needed to execute the first command may be very long, as well as the time for the next two commands, so please be patient when the kernel is being compiled.
After the compilation of our modified kernel, next we can test to show the ctx of each task.

```
	/*
	 * Do this prior waking up the new thread - the thread pointer
	 * might get invalid after that point, if the thread exits quickly.
	 */
	if (!IS_ERR(p)) {
		struct completion vfork;
		/* bingyu shen*/
		p->ctx = 0;
		/* bingyu shen*/
		struct pid *pid;

		trace_sched_process_fork(current, p);

		pid = get_task_pid(p, PIDTYPE_PID);
		nr = pid_vnr(pid);

		if (clone_flags & CLONE_PARENT_SETTID)
			put_user(nr, parent_tidptr);

		if (clone_flags & CLONE_VFORK) {
			p->vfork_done = &vfork;
			init_completion(&vfork);
			get_task_struct(p);
		}

		wake_up_new_task(p);

		/* forking complete and child started to run, tell ptracer */
		if (unlikely(trace))
			ptrace_event_pid(trace, pid);
                                                  1731,14-32    82%
```

Figure 7: Initialize ctx in do_fork function

```
	if (task_on_rq_queued(prev))
		update_rq_clock(rq);

	next = pick_next_task(rq, prev);
	clear_tsk_need_resched(prev);
	clear_preempt_need_resched();
	rq->clock_skip_update = 0;

	if (likely(prev != next)) {
		rq->nr_switches++;
		rq->curr = next;
		++*switch_count;
		/* bingyu shen*/
		next->ctx++;
		/* bingyu shen*/

		trace_sched_switch(preempt, prev, next);
		rq = context_switch(rq, prev, next); /* unlocks the rq */
		cpu = cpu_of(rq);
	} else {
		lockdep_unpin_lock(&rq->lock);
		raw_spin_unlock_irq(&rq->lock);
	}

	balance_callback(rq);
}

static inline void sched_submit_work(struct task_struct *tsk)
{
                                                  3180,18-32    36%
```

Figure 8: Increase ctx by 1

```
static const struct pid_entry tgid_base_stuff[] = {
        DIR("task",       S_IRUGO|S_IXUGO, proc_task_inode_operations, proc_task
_operations),
        DIR("fd",         S_IRUSR|S_IXUSR, proc_fd_inode_operations, proc_fd_ope
rations),
        DIR("map_files",  S_IRUSR|S_IXUSR, proc_map_files_inode_operations, proc
_map_files_operations),
        DIR("fdinfo",     S_IRUSR|S_IXUSR, proc_fdinfo_inode_operations, proc_fd
info_operations),
        DIR("ns",         S_IRUSR|S_IXUGO, proc_ns_dir_inode_operations, proc_ns
_dir_operations),
#ifdef CONFIG_NET
        DIR("net",        S_IRUGO|S_IXUGO, proc_net_inode_operations, proc_net_o
perations),
#endif
        /* bingyu shen */
        ONE("ctx", S_IRUSR, proc_pid_my_file),
        /* bingyu shen */
        REG("environ",    S_IRUSR, proc_environ_operations),
        ONE("auxv",       S_IRUSR, proc_pid_auxv),
        ONE("status",     S_IRUGO, proc_pid_status),
        ONE("personality", S_IRUSR, proc_pid_personality),
        ONE("limits",     S_IRUGO, proc_pid_limits),
#ifdef CONFIG_SCHED_DEBUG
        REG("sched",      S_IRUGO|S_IWUSR, proc_pid_sched_operations),
#endif
#ifdef CONFIG_SCHED_AUTOGROUP
        REG("autogroup",  S_IRUGO|S_IWUSR, proc_pid_sched_autogroup_operations),
#endif
                                                              2759,6        80%
```

Figure 9: Add function entry

```
static const struct dentry_operations tid_map_files_dentry_operations = {
        .d_revalidate   = map_files_d_revalidate,
        .d_delete       = pid_delete_dentry,
};
/* bingyu shen */
static int proc_pid_my_file(struct seq_file *m, struct pid_namespace *ns,
struct pid *pid, struct task_struct *task)
{
        seq_printf(m, "%d\n", task->ctx);
        return 0;
}
/* bingyu shen */
static int proc_map_files_get_link(struct dentry *dentry, struct path *path)
{
        unsigned long vm_start, vm_end;
        struct vm_area_struct *vma;
        struct task_struct *task;
        struct mm_struct *mm;
        int rc;

        rc = -ENOENT;
        task = get_proc_task(d_inode(dentry));
        if (!task)
                goto out;
                                                              1904,16       55%
```

Figure 10: Print seq function

2.2.3 Results

To show the `ctx` of each task, we can write a small test program ctx_test.c to test it.

```
#include <stdio.h>
int main() {
    while (1) getchar();
    return 0;
}
```

Use the following command to compile it.

```
# gcc ctx_test.c -o ctx_test
```

Then execute by

```
# ./ctx_test
```

Fisrt we should get the pid of the process. Open in a new terminal N to execute

```
# ps -e | grep ctx_test
```

Then the pid of ctx_test is 2761. Continue to execute

```
# cd /proc/2761
# cat
```

The result is 1, which is the ctx of process 2761. Then switch to the original terminal to input any char.
Switch to the termial N to execute

```
# cat
```

We find the result is 2, increased by 1. The whole process is shown in Figure.11

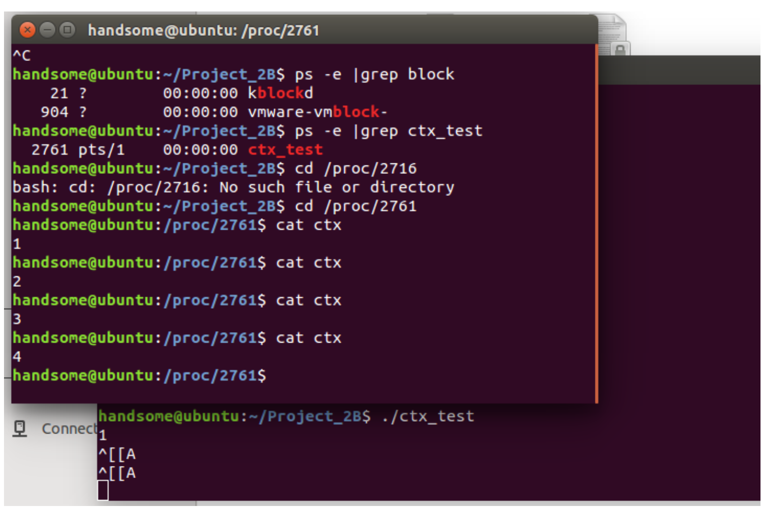


Figure 11: Experiment of Project B

2.2.4 Extention

I think we can use a module to print ctx of all running process. The module's framework is just like the module 3 in the part A. It reads all the running process's name, pid and ctx.

```c
#include <linux/module.h>
#include <linux/list.h>
#include <linux/init.h>
#include <linux/sched.h>
#include <linux/proc_fs.h>
#include <linux/seq_file.h>
MODULE_LICENSE("Dual BSD/GPL");
static int ctx_proc_print(struct seq_file *m, void *v) {
        struct task_struct *task, *p;
        struct list_head *pos;
        int count=0;

        seq_printf(m, "test module init\n");

        task=&init_task;

        list_for_each(pos, &task->tasks)
        {
                p=list_entry(pos, struct task_struct, tasks);
                count++;
                seq_printf(m, "%-15s\t[pid: %d]\t[ctx: %d]\n", p->comm, p->pid,p->ctx);
        }
        seq_printf(m, "Total %d tasks\n", count);

        return 0;
}

static int ctx_proc_open(struct inode *inode, struct file *file) {
        return single_open(file, ctx_proc_print, NULL);
}

static const struct file_operations ctx_proc_fops = {
        .owner = THIS_MODULE,
        .open = ctx_proc_open,
        .read = seq_read,
        .llseek = seq_lseek,
        .release = single_release,
};

static int test_init(void)
{
        proc_create("ctx_proc",0,NULL,&ctx_proc_fops);
```

```
        return 0;
}
static void test_exit(void)
{
        remove_proc_entry("ctx_proc", NULL);
}
module_init(test_init);
module_exit(test_exit);
```

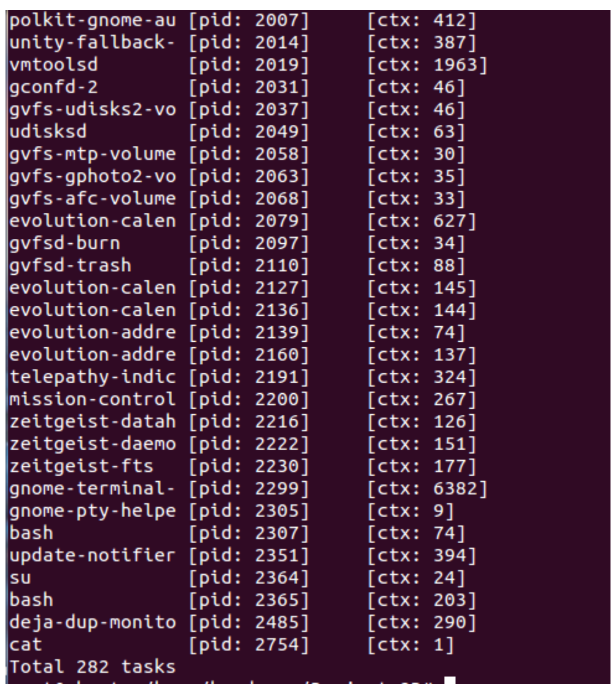
The makefile is like

```
obj-m := ctx_proc.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 execute

```
# make
# insmod ctx_proc.ko
# cat /proc/ctx_proc
```

The result of the command is shown in Figure. 12.



```
polkit-gnome-au [pid: 2007]     [ctx: 412]
unity-fallback- [pid: 2014]     [ctx: 387]
vmtoolsd        [pid: 2019]     [ctx: 1963]
gconfd-2        [pid: 2031]     [ctx: 46]
gvfs-udisks2-vo [pid: 2037]     [ctx: 46]
udisksd         [pid: 2049]     [ctx: 63]
gvfs-mtp-volume [pid: 2058]     [ctx: 30]
gvfs-gphoto2-vo [pid: 2063]     [ctx: 35]
gvfs-afc-volume [pid: 2068]     [ctx: 33]
evolution-calen [pid: 2079]     [ctx: 627]
gvfsd-burn      [pid: 2097]     [ctx: 34]
gvfsd-trash     [pid: 2110]     [ctx: 88]
evolution-calen [pid: 2127]     [ctx: 145]
evolution-calen [pid: 2136]     [ctx: 144]
evolution-addre [pid: 2139]     [ctx: 74]
evolution-addre [pid: 2160]     [ctx: 137]
telepathy-indic [pid: 2191]     [ctx: 324]
mission-control [pid: 2200]     [ctx: 267]
zeitgeist-datah [pid: 2216]     [ctx: 126]
zeitgeist-daemo [pid: 2222]     [ctx: 151]
zeitgeist-fts   [pid: 2230]     [ctx: 177]
gnome-terminal- [pid: 2299]     [ctx: 6382]
gnome-pty-helpe [pid: 2305]     [ctx: 9]
bash            [pid: 2307]     [ctx: 74]
update-notifier [pid: 2351]     [ctx: 394]
su              [pid: 2364]     [ctx: 24]
bash            [pid: 2365]     [ctx: 203]
deja-dup-monito [pid: 2485]     [ctx: 290]
cat             [pid: 2754]     [ctx: 1]
Total 282 tasks
```

Figure 12: Experiment of module solution

3 Memory Management

3.1 Requirements

Write a module that is called mtest. When module loaded, module will create a proc fs entry /proc/mtest. /proc/mtest will accept 3 kind of input.

A. ”listvma” will print all vma of current process in the format of start-addr end-addr permission
e.g
0x10000 0x20000 rwx
0x30000 0x40000 r–

B. ”findpage addr” will find va-¿pa translation of address in current processs mm context and print it. If there is not va-¿pa translation, prink translation not found

C. ”writeval addr val” will change an unsigned long size content in current processs virtual address into val. Note module should write to identity mapping address of addr and verify it from userspace address addr.

All the print can be done with printk and check result with dmesg.

3.2 Background Knowledge

A. Virtual Memory Areas(VMA) and Process

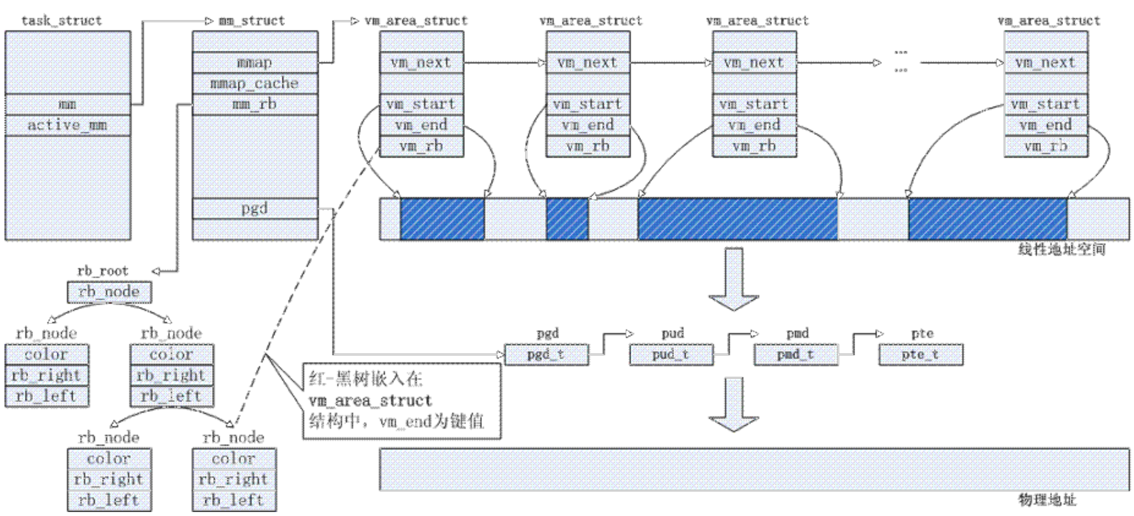


Figure 13: VMA illustration

`struct task_struct` is the process descriptor in Linux, the `mm_struct` field records the VMA of the process. It records the memory related parameters. The prototype of mm_struct is as follows(only list the related codes).

```
struct mm_struct {
    ... ...
    struct vm_area_struct * mmap;/* list of VMAs */
    struct rb_root mm_rb;
```

```
    unsigned long mmap_base;/* base of mmap area */
    unsigned long task_size;/* size of task vm space */

    pgd_t * pgd;
    /* How many users with user space? */
    atomic_t mm_users;
    /* How many references to "struct mm_struct" (users count as 1) */
    atomic_t mm_count;
    int map_count;      /* number of VMAs */
    struct rw_semaphore mmap_sem;
    /* Protects page tables and some counters */
    spinlock_t page_table_lock;
    ...
};
```

Note that mmap_sem is semaphore to lock the mmap, so that if won't be written in two processes at the same time.
`struct vm_area_struct` is the description of virtual memory area. And use mmap to find all VMAs of this process.

```
struct vm_area_struct {
    /* VM area parameters */
    struct mm_struct * vm_mm;
    unsigned long vm_start;
    unsigned long vm_end;

    unsigned short vm_flags;
/* linked list of VM areas per task, sorted by address */
    struct vm_area_struct * vm_next;
    ... ...
};
```

In which the vm_flags is defined as short, the types we used are as follows.

```
 /* currently active flags */
#define VM_READ         0x0001
#define VM_WRITE        0x0002
#define VM_EXEC         0x0004
#define VM_SHARED       0x0008
```

The structure of VMA is illustrated in Figure. 13.

B. Linear Address of Page
The linear page addressing mode is illustrated in Figure. 14.
So from `mm_struct->pgd` we can get the page global directory, which is a physical page frame. This frame contains an array of type pgd_t which is an architecture specific type defined in $< asm/page.h >$. Each active entry in the PGD table points to a page frame containing an array of Page Middle Directory (PMD) entries of type pmd_t which in turn points to page frames containing Page Table Entries (PTE) of type pte_t, which finally points to page frames containing the actual user data.

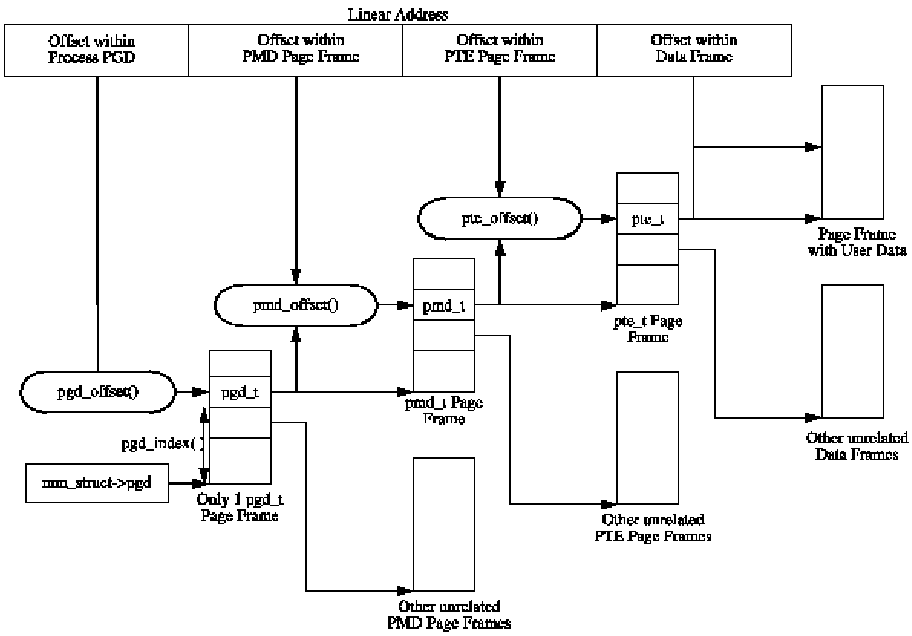


Figure 14: Page Linear Address

The code of get a page address is referenced in $< mm/memory.c >$, the function `follow_page()`. I modify it to make it simpler.

C. VMA to PMA
From virtual memory address to physical memory address, we can find the VMA of a given address first. Then get the specific page with the VMA.
First find the page address of the page. Then add the offset to the page address.

```
kernel_addr = (unsigned long) page_address(page);
kernel_addr += (addr & ~PAGE_MASK);
```

3.3 Experiment

The code of mtest is as follows.

```
#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>
```

```c
#include <linux/fs.h>

static void mtest_dump_vma_list(void)
{
        //get the task_struct of the current process
    struct task_struct *task = current;
    struct mm_struct *mm = task->mm;
        //get the vma area of the current process
    struct vm_area_struct *vma;

    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_EXEC)
            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;
```

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

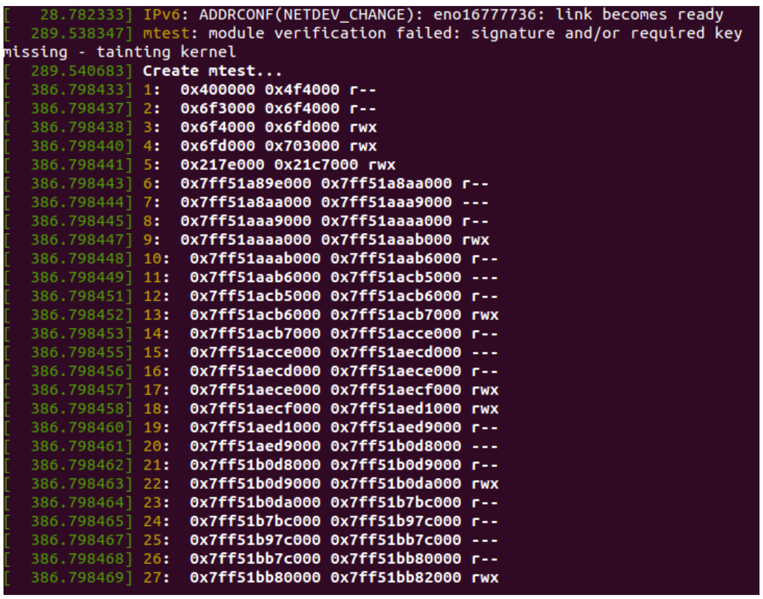
- write a Makefile

```
obj-m := mtest.o
KDIR := /lib/modules/$(shell uname -r)/build

PWD := $(shell pwd)
default:
        $(MAKE) -C $(KDIR) M=$(PWD) modules
clean:
        rm -rf *.o *.ko *.mod.c
```

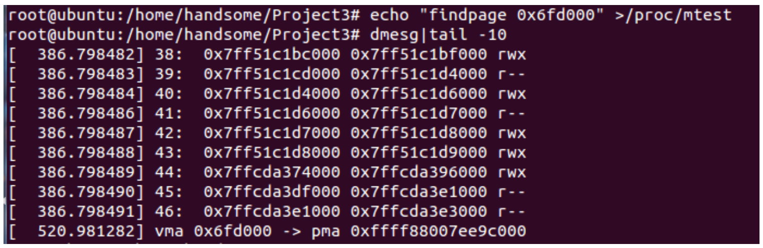
- type ”make” in shell
- type ”sudo insmod mtest.ko”
- type ”sudo su”
- type: echo ”listvma” ¿ /proc/mtest
- type: dmesg (then you will find a lot of vma)

- choose one of them and type : echo ”findpage 0x.........” ¿ /proc/mtest,
  then type : dmesg
  note: you may find that ”translation failed” shows up. But that does not mean you fail the test. Please choose an address between the start and the end of the vma listed. e.g. 0x123 C 0x345, you may want to type: echo findpage 0x300 > /proc/mtest, because you cannot be sure if the beginning of the virtual address is used.
- choose an address that you has rights to write.
  Then type: echo writeval 0x... 123 ¿ /proc/mtest
  type: dmesg
  note: 123 can be any unsigned int
  you’d better choose the same address to test in both step 7 and 8, so that you will see if the physical address is consistent.



```
[   28.782333] IPv6: ADDRCONF(NETDEV_CHANGE): eno16777736: link becomes ready
[  289.538347] mtest: module verification failed: signature and/or required key
missing - tainting kernel
[  289.540683] Create mtest...
[  386.798433] 1:  0x400000 0x4f4000 r--
[  386.798437] 2:  0x6f3000 0x6f4000 r--
[  386.798438] 3:  0x6f4000 0x6fd000 rwx
[  386.798440] 4:  0x6fd000 0x703000 rwx
[  386.798441] 5:  0x217e000 0x21c7000 rwx
[  386.798443] 6:  0x7ff51a89e000 0x7ff51a8aa000 r--
[  386.798444] 7:  0x7ff51a8aa000 0x7ff51aaa9000 ---
[  386.798445] 8:  0x7ff51aaa9000 0x7ff51aaaa000 r--
[  386.798447] 9:  0x7ff51aaaa000 0x7ff51aaab000 rwx
[  386.798448] 10:  0x7ff51aaab000 0x7ff51aab6000 r--
[  386.798449] 11:  0x7ff51aab6000 0x7ff51acb5000 ---
[  386.798451] 12:  0x7ff51acb5000 0x7ff51acb6000 r--
[  386.798452] 13:  0x7ff51acb6000 0x7ff51acb7000 rwx
[  386.798453] 14:  0x7ff51acb7000 0x7ff51acce000 r--
[  386.798455] 15:  0x7ff51acce000 0x7ff51aecd000 ---
[  386.798456] 16:  0x7ff51aecd000 0x7ff51aece000 r--
[  386.798457] 17:  0x7ff51aece000 0x7ff51aecf000 rwx
[  386.798458] 18:  0x7ff51aecf000 0x7ff51aed1000 rwx
[  386.798460] 19:  0x7ff51aed1000 0x7ff51aed9000 r--
[  386.798461] 20:  0x7ff51aed9000 0x7ff51b0d8000 ---
[  386.798462] 21:  0x7ff51b0d8000 0x7ff51b0d9000 r--
[  386.798463] 22:  0x7ff51b0d9000 0x7ff51b0da000 rwx
[  386.798464] 23:  0x7ff51b0da000 0x7ff51b7bc000 r--
[  386.798465] 24:  0x7ff51b7bc000 0x7ff51b97c000 r--
[  386.798467] 25:  0x7ff51b97c000 0x7ff51bb7c000 ---
[  386.798468] 26:  0x7ff51bb7c000 0x7ff51bb80000 r--
[  386.798469] 27:  0x7ff51bb80000 0x7ff51bb82000 rwx
```

Figure 15: listvma experiment



```
root@ubuntu:/home/handsome/Project3# echo "findpage 0x6fd000" >/proc/mtest
root@ubuntu:/home/handsome/Project3# dmesg|tail -10
[  386.798482] 38:  0x7ff51c1bc000 0x7ff51c1bf000 rwx
[  386.798483] 39:  0x7ff51c1cd000 0x7ff51c1d4000 r--
[  386.798484] 40:  0x7ff51c1d4000 0x7ff51c1d6000 rwx
[  386.798486] 41:  0x7ff51c1d6000 0x7ff51c1d7000 r--
[  386.798487] 42:  0x7ff51c1d7000 0x7ff51c1d8000 rwx
[  386.798488] 43:  0x7ff51c1d8000 0x7ff51c1d9000 rwx
[  386.798489] 44:  0x7ffcda374000 0x7ffcda396000 rwx
[  386.798490] 45:  0x7ffcda3df000 0x7ffcda3e1000 r--
[  386.798491] 46:  0x7ffcda3e1000 0x7ffcda3e3000 r--
[  520.981282] vma 0x6fd000 -> pma 0xffff88007ee9c000
```

Figure 16: findpage experiment

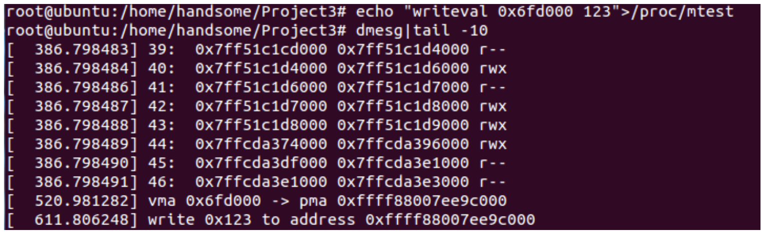


Figure 17: listvma experiment

4 File System

4.1 Requirements

A. Source

- Inode.c/Makefile (kernel source of romfs)
- Test.img (a romfs image, you can mount it to a dir with mount Co loop test.img xxx)
- Say test.img is mounted in t, find t output
  - aa
  - bb
  - ft
  - fo
  - fo/aa

B. Practice 1

- Change romfs code to hide a file/dir with special name
- Test & result
  - insmod romfs hided_file_name= aa
  - Mount Co loop test.img t
  - then ls t, ls t/fo, no ”aa” and ”fo/aa”. found
  - ls t/aa, or ls fo/aa, no found
  - Without the code change, above two operations can find file aa

C. Practice 2

- change the code of romfs to correctly read info of an encrypted romfs
- Test & result
  - insmod romfs encrypted_file_name=bb
  - Mount Co loop test.img t
  - Say bbs original content is bbbbbbb
  - With the change, cat t/bb output ccccccccc

D. Practice 3

- change the code of romfs to add x (execution) bit for a specific file
- Test & result
  - insmod romfs exec_file_name=cc

- Mount Co loop test.img t
- Without code changes ls -l t, output is -rw-r–r–
- With the change, output is -rwxr-xr-x

4.2 Experiment

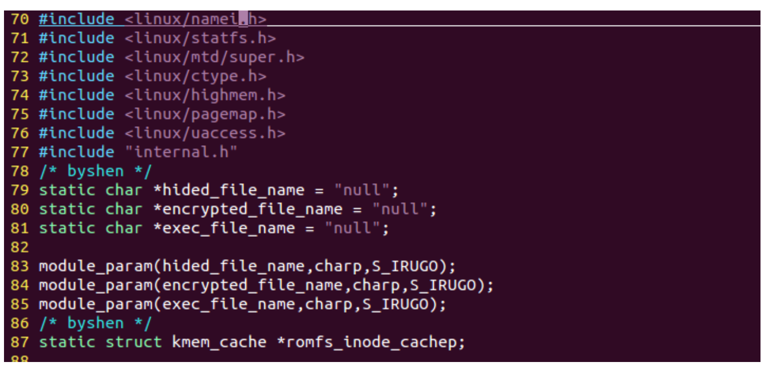
- Create and modofy relevant files

```
# mkdir Project4
# cp -r /usr/src/linux-xxxx/fs/romfs /Project4
# cd romfs
```

- Modify Makefile.

```
obj-m := romfs.o
romfs-y := super.o storage.o
KDIR := /lib/modules/$(shell uname -r)/build
EXTRA_FLAGS := -I(PWD)
PWD := $(shell pwd)
all:
        make -C $(KDIR) M=$(PWD) modules
clean:
        rm *.o *.ko *.mod.c Module.symvers modules.order -f
```

- Modify super.c. There are mainly five modifications.



```
#include <linux/namei.h>
#include <linux/statfs.h>
#include <linux/mtd/super.h>
#include <linux/ctype.h>
#include <linux/highmem.h>
#include <linux/pagemap.h>
#include <linux/uaccess.h>
#include "internal.h"
/* byshen */
static char *hided_file_name = "null";
static char *encrypted_file_name = "null";
static char *exec_file_name = "null";

module_param(hided_file_name,charp,S_IRUGO);
module_param(encrypted_file_name,charp,S_IRUGO);
module_param(exec_file_name,charp,S_IRUGO);
/* byshen */
static struct kmem_cache *romfs_inode_cachep;
```

Figure 18: Modification 1

- create test.img

```
# mkdir test
# genromfs -V "vromfs" -f test.img -d test
```

Then add a directory t to mount the image file.

```
# mkdir t
```

- compile the program.

```
# make
```

```
static int romfs_readpage(struct file *file, struct page *page)
{
    struct inode *inode = page->mapping->host;
    loff_t offset, size;
    unsigned long fillsize, pos;
    void *buf;
    int ret;
    /* byshen */
    int i;
    char *fname = file->f_path.dentry->d_iname;
    /* byshen */

    buf = kmap(page);
    if (!buf)
        return -ENOMEM;

    /* 32 bit warning -- but not for us :) */
    offset = page_offset(page);
    size = i_size_read(inode);
    fillsize = 0;
    ret = 0;
    if (offset < size) {
        size -= offset;
        fillsize = size > PAGE_SIZE ? PAGE_SIZE : size;

        pos = ROMFS_I(inode)->i_dataoffset + offset;

                                                        106,2          15%
```

Figure 19: Modification 2

```

        pos = ROMFS_I(inode)->i_dataoffset + offset;

        ret = romfs_dev_read(inode->i_sb, pos, buf, fillsize);
        if (ret < 0) {
            SetPageError(page);
            fillsize = 0;
            ret = -EIO;
        }
    }
    /* byshen */
    if (strcmp(fname, encrypted_file_name)==0) {
        for (i = 0; i < fillsize; ++i)
            *((char *)buf + i) = 'c';
    }
    /* byshen */
    if (fillsize < PAGE_SIZE)
        memset(buf + fillsize, 0, PAGE_SIZE - fillsize);
    if (ret == 0)
        SetPageUptodate(page);

    flush_dcache_page(page);
    kunmap(page);
    unlock_page(page);
    return ret;
}

static const struct address_space_operations romfs_aops = {
                                                      155,14-17      19%
```

Figure 20: Modification 3

```
        j = romfs_dev_strnlen(i->i_sb, offset + ROMFH_SIZE,
                      sizeof(fsname) - 1);
        if (j < 0)
            goto out;

        ret = romfs_dev_read(i->i_sb, offset + ROMFH_SIZE, fsname, j);
        if (ret < 0)
            goto out;
        fsname[j] = '\0';

        ino = offset;
        nextfh = be32_to_cpu(ri.next);
        /* byshen */
        if (strcmp(fsname, hided_file_name)==0) {
            offset = nextfh & ROMFH_MASK;
            continue;
        }
        /* byshen */
        if ((nextfh & ROMFH_TYPE) == ROMFH_HRD)
            ino = be32_to_cpu(ri.spec);
        if (!dir_emit(ctx, fsname, j, ino,
                romfs_dtype_table[nextfh & ROMFH_TYPE]))
            goto out;

        offset = nextfh & ROMFH_MASK;
    }
out:
    return 0;
                                                    213,6-12      30%
```

Figure 21: Modification 4

```

    /* byshen */
    if (strcmp(name, exec_file_name)==0) {
        inode->i_mode |= S_IXUGO;
    }
    /* byshen */

    if (IS_ERR(inode)) {
        ret = PTR_ERR(inode);
        goto error;
    }
    goto outi;

    /*
     * it's a bit funky, _lookup needs to return an error code
     * (negative) or a NULL, both as a dentry.  ENOENT should not
     * be returned, instead we need to create a negative dentry by
                                                    282,5-8       40%
```

Figure 22: Modification 5

4.3 Results

Before modification.

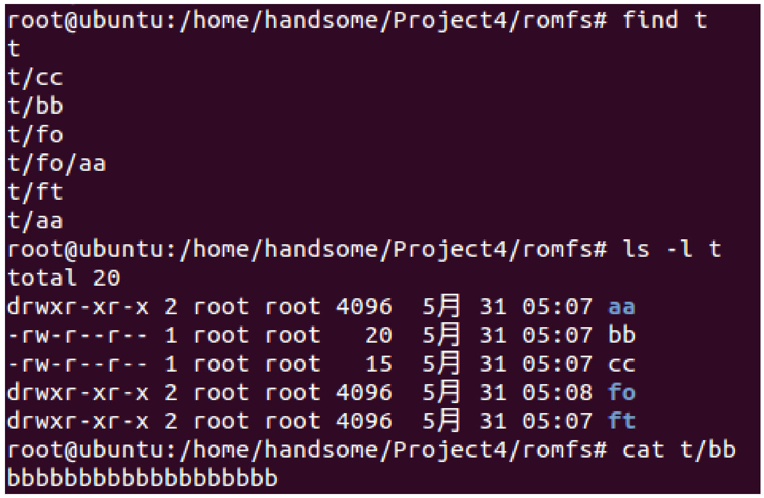


Figure 23: Practice 1-3 before modification

I write a run.sh to simplify the test.

```
echo "--- Practice 1 ---"
insmod romfs.ko hided_file_name=aa
mount -o loop test.img t
find t
umount test.img
rmmod romfs.ko
echo "--- Practice 2 ---"
insmod romfs.ko encrypted_file_name=bb
mount -o loop test.img t
cat t/bb
umount test.img
rmmod romfs.ko
echo " "
echo "--- Practice 3 ---"
insmod romfs.ko exec_file_name=cc
mount -o loop test.img t
ls -l t
umount test.img
rmmod romfs.ko
```

And the test result after modification is correspondingly as Figure.24.

5 Conclusion

In the projects of this lecture, I finished four parts of the project. The first project requires to upgrade the linux kernel to the newest version; the second project part A focus on writing

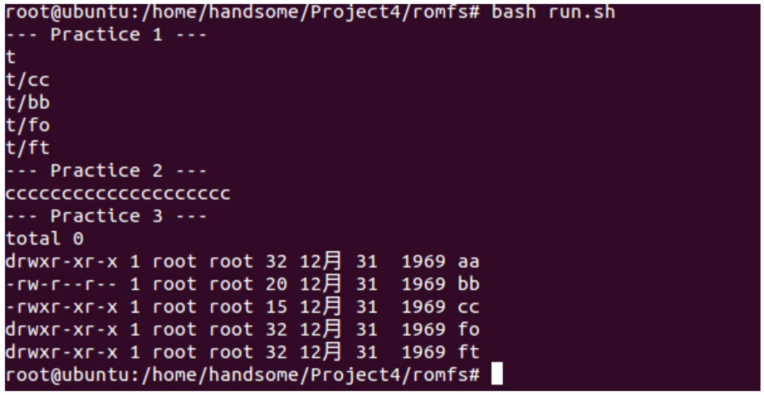


Figure 24: Practice 1-3 after modification

module program and install it in the kernel program, part B focus on process management. which requires us to add a member in the task struct and record how many times it has been executed. The third project focus on memory management, which requires to write a module mtestand can write andfind certain memory address of all running process. Thefourth project focus on file system, in which we can learn a lot about the romfs file system. All programs test runs on Unbuntu 15.10 and results are checked by TA. The source and modified program are attached.

The experiment promote my understanding of the linux proc, memory and file system, and I gain a lot of experiments in reading the source code and kernel programming.

Finally, I appreciate the financial support from Prof. Wu and TA that helped the project.

6 References

[1] Linux kernel experiment handbook of SJTU CS353
[2] CS353 Course materials