Project 3 Memory management

- 5130309717
- 朱文豪

Target

- 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
    - “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—
    - “findpage addr” will find va->pa translation of address in current process’s mm context and print it. If there is not va->pa translation, prink “tranlsation not found”
    - “writeval addr val” will change an unsigned long size content in currentprocess’s 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.

1. Add mtest module

All process are based on struct `task_strcut`, we can get or modify its member.

```
#include <linux/module.h> /* Needed by all modules */
#include <linux/kernel.h> /* Needed for KERN_INFO */
#include <linux/proc_fs.h>
#include <linux/seq_file.h> /* Needed for seq interface */
#include <linux/sched.h> /* Needed for task_struct*/
#include <linux/mm.h> /* Needed for vm_flags */
#include <asm/uaccess.h> /* Needed for copy_from_user */
#include <asm/io.h> /* Needed for addr convert*/

#define proc_name  "mtest"
```

```
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Memory Management Homework");
MODULE_AUTHOR("weehowe.z@gmail.com");

char buf[128];/* proc file content */

struct proc_dir_entry *entry;

static void listvma(unsigned int pid){

    struct task_struct *task;
    struct mm_struct *mm;
    struct vm_area_struct *vma; /*this struct defines a memory VMM memory
area*/

    if (pid == 0){
        task = current; /* get the task_struct of the current process */
    }   else {
        task = pid_task(find_get_pid(pid),PIDTYPE_PID); /* or get the
task_struct by pid */
        if (!task){
            printk(KERN_ALERT "Error: Invalid pid %d, no such process\n",
pid);
            return;
        }
    }
    mm = task->mm;
    vma = mm->mmap ;

    // down_read(&mm->mmap_sem);/* process protect */

    printk(KERN_INFO "start-addr\tend-addr\tpermission\n");
    while (vma){
        char vma_perm[4];
        //handle premissions
        if (vma->vm_flags & VM_READ) vma_perm[0] = 'r';
            else vma_perm[0] = '-';
        if (vma->vm_flags & VM_WRITE) vma_perm[1] = 'w';
            else vma_perm[1] = '-';
        if (vma->vm_flags & VM_EXEC) vma_perm[2] = 'x';
            else vma_perm[2] = '-';
        vma_perm[3] = '\0';

        printk(KERN_INFO "0x%lx\t0x%lx\t%s\n",vma->vm_start,vma-
>vm_end,vma_perm);

        vma = vma->vm_next;
    }
    // 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))) return NULL;

    pud = pud_offset(pgd, addr);   //get pud
    if (pud_none(*pud) || unlikely(pud_bad(*pud))) return NULL;

    pmd = pmd_offset(pud, addr);   //get pmd
    if (pmd_none(*pmd) || unlikely(pmd_bad(*pmd))) return NULL;

    pte = pte_offset_map_lock(mm, pmd, addr, &ptl); //get pte
    if (!pte || !pte_present(*pte) ) return NULL;

    page = pfn_to_page(pte_pfn(*pte));
        if (!page) return NULL;

    get_page(page);

    // pte_unmap_unlock(pte, ptl);
    return page;
}

static void findpage(unsigned long addr){

    struct task_struct *task = current;
    struct mm_struct *mm = task->mm;
    struct vm_area_struct *vma = mm->mmap;

    struct page *page;

    unsigned long phy_addr;

    page = my_follow_page(vma, addr);
    if (!page)
    {
        printk("Error: Translation failed.\n");
        return;
    }

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

    // unsigned long phy_addr = virt_to_phys((void *)addr);
    printk(KERN_INFO "Virtual Address 0x%lx -> Physical Address
0x%lx\n",addr ,phy_addr);

}
```

```
static void wirteval(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 phy_addr;

    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(KERN_ALERT "Cannot write to 0x%lx, permission denied\n",
addr);
                        return;
        }
        page = my_follow_page(vma, addr);
        if (!page)
        {
            printk("page not found 0x%lx\n", addr);
            return;
        }

        phy_addr = (unsigned long) page_address(page);
        phy_addr += (addr &~ PAGE_MASK);
        printk("write 0x%lx to physical address 0x%lx\n", val, phy_addr);
        *(unsigned long *)phy_addr = val;
        put_page(page);
    }
    else{
        printk(KERN_ALERT "Virtual Address 0x%lx not valid\n", addr);
    }
}

static int proc_show(struct seq_file *m, void *v) { printk(KERN_INFO
"/proc/%s read called\n", proc_name);
    seq_printf(m, "%s", buf);
    return 0;
}

static int proc_open(struct inode *inode, struct  file *file) {
    return single_open(file, proc_show, NULL);
}

static ssize_t proc_write(struct file *file, const char __user *buffer,
size_t count, loff_t *data){

    unsigned int pid;
    unsigned long addr, val;
```

```
    printk(KERN_INFO "/proc/%s write called\n", proc_name);
    memset(buf,0,sizeof(buf)/sizeof(char));//clear the buffer

    if (count > sizeof(buf)) return -EINVAL;
    if (copy_from_user(buf, buffer, count)) return -EFAULT;

    if (memcmp(buf,"listvma",7) == 0) {
        printk(KERN_INFO "/proc/%s listvma called\n", proc_name);

        if (count == 8) listvma(0);/*no pid args*/
        else if (sscanf(buf+7, "%d", &pid) == 1){
            listvma(pid);
        }
        else return -EFAULT;
    }

    else if (memcmp(buf,"findpage",8) == 0){
        printk(KERN_INFO "/proc/%s findpage called\n", proc_name);

        if (sscanf(buf+8, "%lx", &addr) == 1){
            printk(KERN_INFO "%lx\n", addr);
            findpage(addr);
        }
        else return -EFAULT;
    }

    else if (memcmp(buf,"writeval",8) == 0){
        printk(KERN_INFO "/proc/%s writeval called\n", proc_name);
        if (sscanf(buf+8, "%lx %lx", &addr, &val) == 2){
            printk(KERN_INFO "%lx %lx\n", addr, val);
            wirteval(addr,val);
        }
        else return -EFAULT;
    }

    return count;
}

static const struct file_operations proc_fops = {
    .owner = THIS_MODULE,
    .open = proc_open,
    .read = seq_read,
    .write = proc_write,
    .llseek = seq_lseek,
    .release = single_release,
};

int init_module()
{
    entry = proc_create(proc_name, 0644, NULL, &proc_fops);

    if (!entry) {
        remove_proc_entry(proc_name, NULL);
        printk(KERN_ALERT "Error: Could not initialize
/proc/%s\n",proc_name);
```

```
        return -ENOMEM;
    }
    printk(KERN_INFO "------------\n");
    printk(KERN_INFO "/proc/%s created\n", proc_name);
    return 0;
}

void cleanup_module()
{
    remove_proc_entry(proc_name,NULL);
    printk(KERN_INFO "/proc/%s removed\n", proc_name);
    printk(KERN_INFO "------------\n");
}
```

2. Test

create a little program `test.c`

```
#include <stdio.h>
#include <string.h>

int main()
{
    int i = 0;
    int v = 0;
    char s[100];
    FILE *mtest;
    printf("Virtual Memory address of num: %p\n", &v);
    printf("Current num value %d\n", v);

    while(true){

        mtest = fopen("/proc/mtest", "w+");

        fgets(s, 80, stdin);

        if(memcmp(s, "exit", 4) == 0) break;

        else if(memcmp(s, "print", 5) == 0) {
            printf("Current num value %d\n", v);
        }

        else if(memcmp(s, "listvma", 7) == 0) {
            printf("'listvma' called, see dmesg\n");
            fprintf(mtest, "%s\n", "listvma");
        }

        else if(memcmp(s, "write", 5) == 0){
            sscanf(s, "write %d", &i);
            printf("Excute writeval 0x%lx %d\n", (unsigned long)(void*)&v,
```

```
i);
            fprintf(mtest, "writeval 0x%lx %lx\n", (unsigned long)(void*)&v,
(unsigned long)i);
        }

        fclose(mtest);
    }
    return 0;
}
```

Create a test scripts for easy test

```
#!/bin/sh
rmmod mtest

echo "------------"

make
insmod mtest.ko
cat /proc/mtest
echo "listvma" > /proc/mtest
# cat /proc/mtest
echo "listvma 9181" > /proc/mtest
# cat /proc/mtest
echo "findpage 0x400000" > /proc/mtest
# cat /proc/mtest
echo "writeval 0x7fffbda9d000 0x1" > /proc/mtest
# cat /proc/mtest
dmesg -c
```

run test

```
$ sudo chmod +x test.sh
$ ./test.sh
```