Navid Malek
navidmalekedu@gmail.com
Project Phase 2

Operating Systems
Spring 2017

Part 3

Implementation of desired kernel module and user program

Implement the module

Now What will this module do and how?
tops_init is the first routin that will be runned , the module is well commented so just read the comments to see what happens.

```
#include <linux/init.h>
#include <linux/module.h> //for module programming
#include <linux/sched.h> //for task_struct
#include <linux/jiffies.h> //for file_operations write and read
#include <linux/kernel.h> //for kernel programming
#include <linux/cred.h>
#include <linux/proc_fs.h> //for using proc
#include <linux/seq_file.h> // for using seq operations
#include <linux/fs.h>   //for using file_operations
#include <linux/mm_types.h> //for using vm_area struct
#include <asm/uaccess.h>    //for user to kernel and vice versa access
#include <linux/string.h> //for string libs

MODULE_LICENSE("Dual BSD/GPL"); //module license


static char buff[20]="1"; //the common(global) buffer between kernel and user
space
static int user_pid;    //the desired pid that we get from user
static int numberOpens = 0; //number of opens(writes) to the pid file

//skip these instances (will be described bellow)
static struct proc_dir_entry *topsDir, *topsFile, *topsWrite;

static int procfile_open(struct inode *inode, struct file *file);
static ssize_t procfile_read(struct file*, char*, size_t, loff_t*);
static ssize_t procfile_write(struct file*, const char*, size_t, loff_t*);


//det proc file_operations starts

//this function is the base function to gather information from kernel
static int tops_show(struct seq_file *m, void *v) {
        struct task_struct *task;
        u64 delta, total;
        u64 usage; //for knowing cpu usage
        sscanf(buff, "%d", &user_pid);  //type cast pid from user(buff) to
integer
        task = pid_task(find_vpid(user_pid), PIDTYPE_PID);  //get the task from
pid

        total = ((task->utime + task->stime) / HZ) * 1000000000;
        delta = ktime_get_ns() - task->start_time;
        usage = (1000 * total) / delta; //cpu usage

        //now print the information we want to the det file
        seq_printf(m, "PID \tNAME \tCPU \tSTART_CODE \tEND_CODE
\tSTART_DATA\tEND_DATA \tBSS_START\tBSS_END\n");
        seq_printf(m, "%.5d\t%.7s\t%llu.%llu\t0x%.13lx\t0x%.13lx\t0x%.13lx\t0x
%.13lx\t0x%.13lx\t0x%.13lx\n", task->pid,task->comm,usage / 10, usage % 10,
task->active_mm->start_code, task->active_mm->end_code,task->active_mm-
>start_data,task->active_mm->end_data,task->active_mm->mmap->vm_next->vm_next-
>vm_start,task->active_mm->mmap->vm_next->vm_next->vm_end);

        return 0;
}

//runs when openning file
static int tops_open(struct inode *inode, struct file *file) {
        return single_open(file, tops_show, NULL); //calling tops_show
}

//file operations of det proc
static const struct file_operations tops_fops = {
        .owner = THIS_MODULE,
        .open = tops_open, //this is really important!
        .read = seq_read,
        .llseek = seq_lseek,
        .release = single_release,
};




//elf proc file_operations starts

//runs when elf opens
//will be called every time this file is accessed shows number of accessed
times
static int procfile_open(struct inode *inode, struct file *file)
{
        numberOpens++;
        printk(KERN_INFO "procfile opened %d times", numberOpens);
        return 0;
}

//when we cat elf file this function will be runned (this is useless here)
because our info is in det file not here!
static ssize_t procfile_read(struct file *file, char *buffer, size_t length,
loff_t *offset)
```

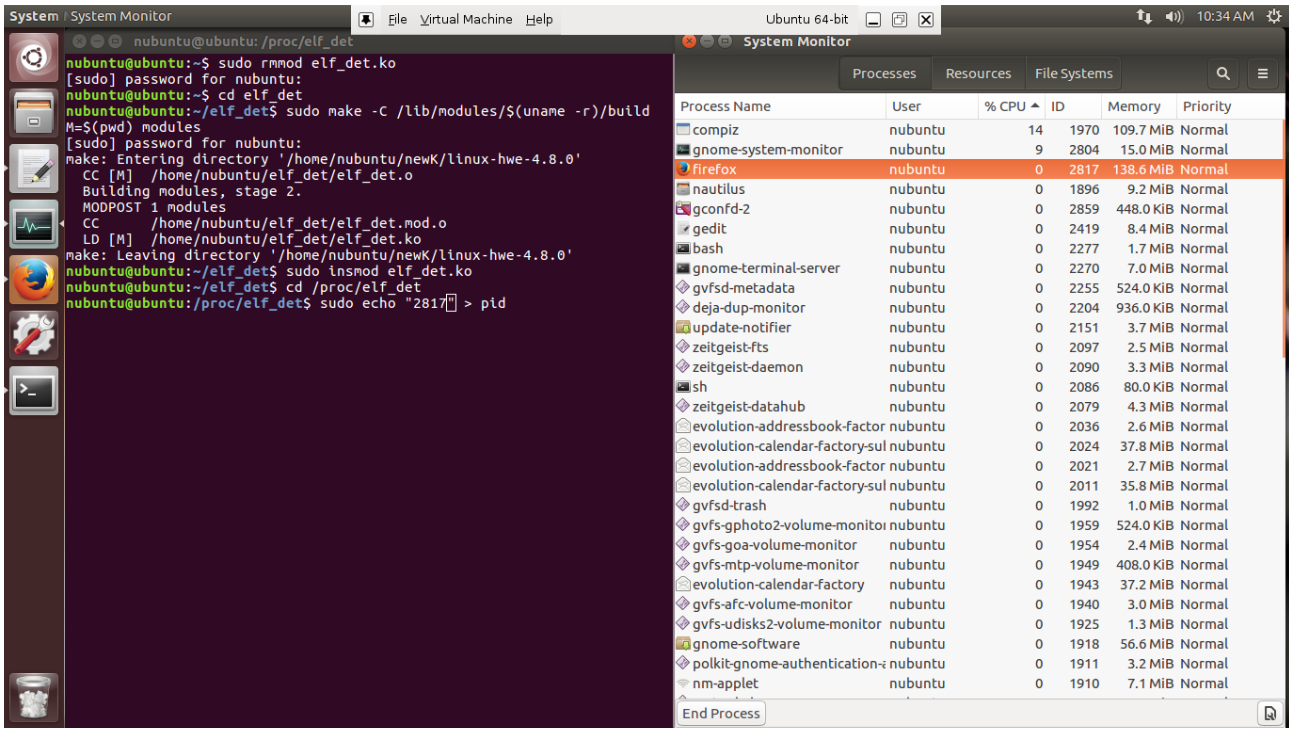
```
{
    static int finished = 0; //normal return value other than '0' will cause
loop
    int ret = 0;

    printk(KERN_INFO "procfile read called\n");

    if (finished) {
        printk(KERN_INFO "procfs read: END\n");
        finished = 0;
        return 0;
    }

    finished = 1;
    ret = sprintf(buffer, "buff variable : %s\n", buff);
    return ret;
}

//most important function of elf! called when we write some charachters into it
static ssize_t procfile_write(struct file *file, const char *buffer, size_t
length, loff_t *offset)
{
        strncpy_from_user(buff, buffer, sizeof(buff)); //copy the charachters
to buff (global buffer, inorder to use it in kernel)
        printk(KERN_INFO "procfs_write called  called\n");
        return -EFAULT; // (return 0;) will result a loop for unknown reason.
same solution as procfile_read did not work
}


static struct file_operations write_fops = {
    .owner = THIS_MODULE,
    .open = procfile_open,
    .read = procfile_read,
    .write = procfile_write,//this is the important part
};


static int tops_init(void) {
        topsDir = proc_mkdir("elf_det", NULL); //creating the directory:
elf_det in proc

        if (!topsDir) {
                return -ENOMEM;
        }
        //0777 means full premmisions for the file
        topsFile = proc_create("det", 0777, topsDir, &tops_fops); //create proc
file det with tops_fops file operations
        printk("det initiated; /proc/elf_det/det created\n");
        topsWrite = proc_create("pid",0777, topsDir, &write_fops);////create
proc file pid with write_fops file operations
        printk("pid initiated; /proc/elf_det/pid created\n");

        if (!topsFile) {
                return -ENOMEM;
```

Navid Malek
navidmalekedu@gmail.com
Project Phase 2

Operating Systems
Spring 2017

```
        }


        return 0;
}

//the remove operations done by module(cleaning up)
static void tops_exit(void) {
        proc_remove(topsFile);
        printk("tops exited; /proc/elf_det/det deleted\n");
        proc_remove(topsWrite);
        printk("tops exited; /proc/elf_det/pid deleted\n");
        proc_remove(topsDir);
}

//macros for init and exit
module_init(tops_init);
module_exit(tops_exit);
```

Ok now lets use the kernel module to see it in action:



```
nubuntu@ubuntu:~$ sudo rmmod elf_det.ko
[sudo] password for nubuntu:
nubuntu@ubuntu:~$ cd elf_det
nubuntu@ubuntu:~/elf_det$ sudo make -C /lib/modules/$(uname -r)/build
M=$(pwd) modules
[sudo] password for nubuntu:
make: Entering directory '/home/nubuntu/newK/linux-hwe-4.8.0'
  CC [M]  /home/nubuntu/elf_det/elf_det.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/nubuntu/elf_det/elf_det.mod.o
  LD [M]  /home/nubuntu/elf_det/elf_det.ko
make: Leaving directory '/home/nubuntu/newK/linux-hwe-4.8.0'
nubuntu@ubuntu:~/elf_det$ sudo insmod elf_det.ko
nubuntu@ubuntu:~/elf_det$ cd /proc/elf_det
nubuntu@ubuntu:/proc/elf_det$ sudo echo "2817" > pid
```

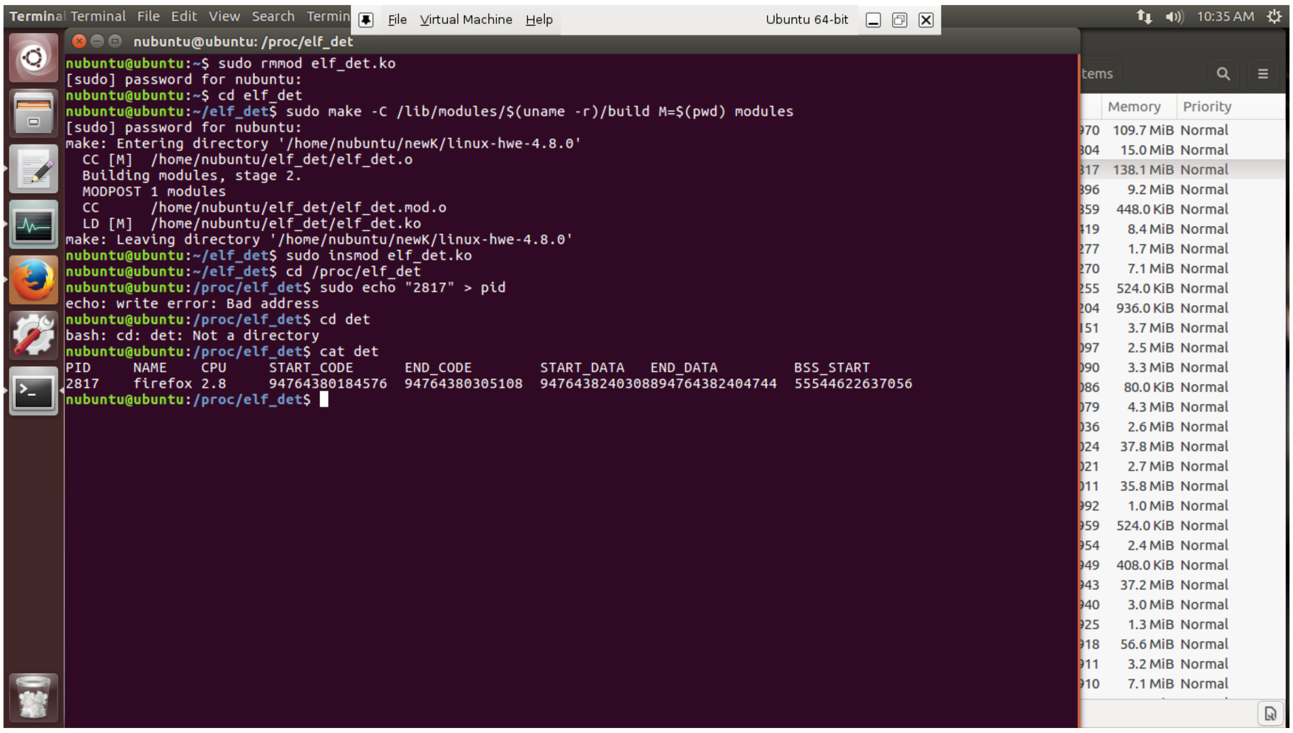
Process Name	User	% CPU	ID	Memory	Priority
compiz	nubuntu	14	1970	109.7 MiB	Normal
gnome-system-monitor	nubuntu	9	2804	15.0 MiB	Normal
firefox	nubuntu	0	2817	138.6 MiB	Normal
nautilus	nubuntu	0	1896	9.2 MiB	Normal
gconfd-2	nubuntu	0	2859	448.0 KiB	Normal
gedit	nubuntu	0	2419	8.4 MiB	Normal
bash	nubuntu	0	2277	1.7 MiB	Normal
gnome-terminal-server	nubuntu	0	2270	7.0 MiB	Normal
gvfsd-metadata	nubuntu	0	2255	524.0 KiB	Normal
deja-dup-monitor	nubuntu	0	2204	936.0 KiB	Normal
update-notifier	nubuntu	0	2151	3.7 MiB	Normal
zeitgeist-fts	nubuntu	0	2097	2.5 MiB	Normal
zeitgeist-daemon	nubuntu	0	2090	3.3 MiB	Normal
sh	nubuntu	0	2086	80.0 KiB	Normal
zeitgeist-datahub	nubuntu	0	2079	4.3 MiB	Normal
evolution-addressbook-factor	nubuntu	0	2036	2.6 MiB	Normal
evolution-calendar-factory-sul	nubuntu	0	2024	37.8 MiB	Normal
evolution-addressbook-factor	nubuntu	0	2021	2.7 MiB	Normal
evolution-calendar-factory-sul	nubuntu	0	2011	35.8 MiB	Normal
gvfsd-trash	nubuntu	0	1992	1.0 MiB	Normal
gvfs-gphoto2-volume-monitoı	nubuntu	0	1959	524.0 KiB	Normal
gvfs-goa-volume-monitor	nubuntu	0	1954	2.4 MiB	Normal
gvfs-mtp-volume-monitor	nubuntu	0	1949	408.0 KiB	Normal
evolution-calendar-factory	nubuntu	0	1943	37.2 MiB	Normal
gvfs-afc-volume-monitor	nubuntu	0	1940	3.0 MiB	Normal
gvfs-udisks2-volume-monitor	nubuntu	0	1925	1.3 MiB	Normal
gnome-software	nubuntu	0	1918	56.6 MiB	Normal
polkit-gnome-authentication-a	nubuntu	0	1911	3.2 MiB	Normal
nm-applet	nubuntu	0	1910	7.1 MiB	Normal

as you see 2817 is the pid of firefox, so we will first write 2817 to the pid proc file.

Navid Malek
navidmalekedu@gmail.com
Project Phase 2

Operating Systems
Spring 2017

Now lets read the det file to see the information:


Terminal Terminal File Edit View Search Termin
File Virtual Machine Help
Ubuntu 64-bit
10:35 AM
nubuntu@ubuntu: /proc/elf_det
tems

```
nubuntu@ubuntu:~$ sudo rmmod elf_det.ko
[sudo] password for nubuntu:
nubuntu@ubuntu:~$ cd elf_det
nubuntu@ubuntu:~/elf_det$ sudo make -C /lib/modules/$(uname -r)/build M=$(pwd) modules
[sudo] password for nubuntu:
make: Entering directory '/home/nubuntu/newK/linux-hwe-4.8.0'
  CC [M]  /home/nubuntu/elf_det/elf_det.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/nubuntu/elf_det/elf_det.mod.o
  LD [M]  /home/nubuntu/elf_det/elf_det.ko
make: Leaving directory '/home/nubuntu/newK/linux-hwe-4.8.0'
nubuntu@ubuntu:~/elf_det$ sudo insmod elf_det.ko
nubuntu@ubuntu:~/elf_det$ cd /proc/elf_det
nubuntu@ubuntu:/proc/elf_det$ sudo echo "2817" > pid
echo: write error: Bad address
nubuntu@ubuntu:/proc/elf_det$ cd det
bash: cd: det: Not a directory
nubuntu@ubuntu:/proc/elf_det$ cat det
PID     NAME     CPU     START_CODE      END_CODE        START_DATA    END_DATA          BSS_START
2817    firefox 2.8     94764380184576  94764380305108  9476438240308894764382404744  55544622637056
nubuntu@ubuntu:/proc/elf_det$ 
```

	Memory	Priority
970	109.7 MiB	Normal
304	15.0 MiB	Normal
317	138.1 MiB	Normal
396	9.2 MiB	Normal
359	448.0 KiB	Normal
419	8.4 MiB	Normal
277	1.7 MiB	Normal
270	7.1 MiB	Normal
255	524.0 KiB	Normal
204	936.0 KiB	Normal
151	3.7 MiB	Normal
097	2.5 MiB	Normal
090	3.3 MiB	Normal
086	80.0 KiB	Normal
079	4.3 MiB	Normal
036	2.6 MiB	Normal
024	37.8 MiB	Normal
021	2.7 MiB	Normal
011	35.8 MiB	Normal
992	1.0 MiB	Normal
959	524.0 KiB	Normal
954	2.4 MiB	Normal
949	408.0 KiB	Normal
943	37.2 MiB	Normal
940	3.0 MiB	Normal
925	1.3 MiB	Normal
918	56.6 MiB	Normal
911	3.2 MiB	Normal
910	7.1 MiB	Normal

everything is perfect!

(edit : %lx in the code prints in hex format, the screenshot shows in decimal format (%lu),

also screenshot supports BSS_START)

Navid Malek
navidmalekedu@gmail.com
Project Phase 2

Operating Systems
Spring 2017

Implement user program

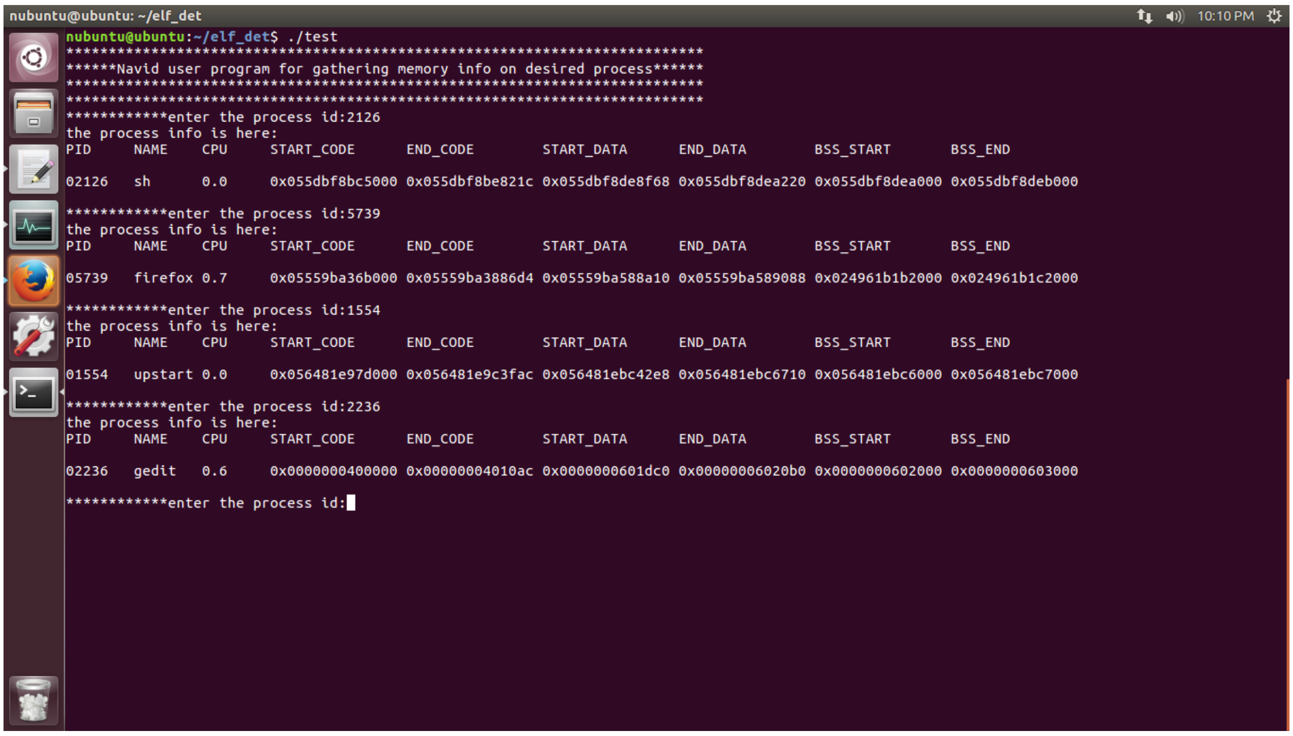
Just do the writing and reading from file automatically!

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

int main()
{
    FILE *fp;
    char pid_user[20];
    char buff[2048];


printf("************************************************************************
****\n");
    printf("******Navid user program for gathering memory info on desired
process******\n");

printf("************************************************************************
****\n");

printf("************************************************************************
****\n");
while(1==1){
    printf("************enter the process id:");
    scanf("%s",pid_user);

    fp = fopen("/proc/elf_det/pid","w");
    fprintf(fp,"%s", pid_user);
    fclose(fp);

    printf("the process info is here:\n");
    fp = fopen("/proc/elf_det/det","r");
    fgets(buff, 2048, (FILE*)fp);
    printf("%s\n",buff);
    fgets(buff, 2048, (FILE*)fp);
    printf("%s\n",buff);
    fclose(fp);
}
return 0;
}
```

Running the user program:

Navid Malek
navidmalekedu@gmail.com
Project Phase 2

Operating Systems
Spring 2017



```
nubuntu@ubuntu:~/elf_det$ ./test
***************************************************************************
******Navid user program for gathering memory info on desired process******
***************************************************************************
***************************************************************************
************enter the process id:2126
the process info is here:
PID     NAME    CPU     START_CODE      END_CODE        START_DATA      END_DATA        BSS_START       BSS_END

02126   sh      0.0     0x055dbf8bc5000 0x055dbf8be821c 0x055dbf8de8f68 0x055dbf8dea220 0x055dbf8dea000 0x055dbf8deb000

************enter the process id:5739
the process info is here:
PID     NAME    CPU     START_CODE      END_CODE        START_DATA      END_DATA        BSS_START       BSS_END

05739   firefox 0.7     0x05559ba36b000 0x05559ba3886d4 0x05559ba588a10 0x05559ba589088 0x024961b1b2000 0x024961b1c2000

************enter the process id:1554
the process info is here:
PID     NAME    CPU     START_CODE      END_CODE        START_DATA      END_DATA        BSS_START       BSS_END

01554   upstart 0.0     0x056481e97d000 0x056481e9c3fac 0x056481ebc42e8 0x056481ebc6710 0x056481ebc6000 0x056481ebc7000

************enter the process id:2236
the process info is here:
PID     NAME    CPU     START_CODE      END_CODE        START_DATA      END_DATA        BSS_START       BSS_END

02236   gedit   0.6     0x0000000400000 0x00000004010ac 0x0000000601dc0 0x00000006020b0 0x0000000602000 0x0000000603000

************enter the process id:
```

very well!!

End of part 3

If you have any questions you can contact me through email.

Best wishes

navid malek.