CS 550: Program 2

Zachary Seymour

March 30, 2014

1 Configuring and compiling the Linux Kernel

For compiling the kernel, I mainly followed the instructions provided in the assignment with little change. I ran into a few issues when setting up via `make menuconfig`. In particular, I at first removed the NFS filesystem, which made the system unable to boot, probably an issue with the PXE boot system. Once I had resolved this issue and had a successful boot to log-in, I had no working mouse or keyboard. From there, I learned about UHCI, added the necessary modules, recompiled and rebooted successfully.

2 System calls

```
asmlinkage int my_xtime(struct timespec *current_time)
{
        struct timespec now;
        int ret = 0;
        //if(!access_ok(VERIFY_WRITE, current_time, sizeof(now)))
        //      return -EFAULT;
        getnstimeofday(&now);
        ret = copy_to_user(current_time, &now, sizeof(now));
        printk(KERN_ALERT "It is now %ld nanoseconds.\n", now.tv_nsec);
        return ret;
}
EXPORT_SYMBOL(my_xtime);
```

While implementing the required system call, I found that the `timekeeper` struct no longer had the `xtime` member, as of around March 2012. From there, I found the `getnstimeofday(struct timespec)` function, which allowed me to fill a `struct timespec` and then copy it to user-level memory.

My code for this can be seen above, and my user-level program to test it follows.

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <linux/unistd.h>
#define __NR_my_xtime 314

int main() {
   int ret;
   struct timespec *current_time = malloc(sizeof(struct timespec));
   ret = syscall(__NR_my_xtime, current_time);
   if(ret==0)
      printf("Time is %ld ns.\n", current_time->tv_nsec);
   else
      printf("ret = %d, errno = %d\n", ret, errno);
   return 0;
}
```

3 Kernel Module

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

static int xtime_proc_show(struct seq_file *m, void *v) {
  struct timespec now;
  getnstimeofday(&now);
  seq_printf(m, "%ld %ld\n", now.tv_sec, now.tv_nsec);
  return 0;
}

static int xtime_proc_open(struct inode *inode, struct  file *file) {
  return single_open(file, xtime_proc_show, NULL);
}

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

static int __init xtime_proc_init(void) {
  proc_create("myxtime", 0, NULL, &proc_fops);
  return 0;
}

static void __exit xtime_proc_exit(void) {
  remove_proc_entry("myxtime", NULL);
}

MODULE_LICENSE("GPL");
module_init(xtime_proc_init);
module_exit(xtime_proc_exit);
```

Implementing my module was fairly straight forward. I found a few tutorials online, which led to me finding the `single-open` and `seq_printf` functions, which were key to writing to my file. From there, it was merely an matter of filling out the `struct timespec` again and printing the two values. Once again, my code for the module is above, with my user-level test below.

```
#define N 50

#include <sys/time.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
        struct timeval gtodTimes[N];
        char *procClockTimes[N];
        FILE  *my_xtime;
        for(int i = 0; i < N; i++)
        {
                my_xtime = fopen("/proc/myxtime", "r");
                if(my_xtime == NULL)
                {
                        printf("File pointer null.");
                        return 1;
                }
                /* printf("Iteration %d.\n", i);  */
                gettimeofday(&gtodTimes[i], 0);
                procClockTimes[i] = malloc(sizeof(char)*22);
                fgets(procClockTimes[i], 22, my_xtime);
                fclose(my_xtime);
        }
        //printf("Now outputting results.\n");
        for(int i = 0; i < N; i++)
        {
                printf("%ld %ld | %s\n",
                       gtodTimes[i].tv_sec,gtodTimes[i].tv_usec,
                       procClockTimes[i]);
                free(procClockTimes[i]);
        }
}
```

4 Experimenting with “Bad” Code

I tried several of the suggested experiments to crash the module.

4.1 Using Library Functions

First, I tried to use a standard C library function, as seen below, but the module would not even compile.

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

static int xtime_proc_show(struct seq_file *m, void *v) {
  struct timespec now;
  getnstimeofday(&now);
  //Using printf instead of printk
  seq_printf(m, "%ld %ld\n", now.tv_sec, now.tv_nsec);
  printf("Module is working.");
  return 0;
}

static int xtime_proc_open(struct inode *inode, struct  file *file) {
  return single_open(file, xtime_proc_show, NULL);
}

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

static int __init xtime_proc_init(void) {
  proc_create("myxtime", 0, NULL, &proc_fops);
  return 0;
}

static void __exit xtime_proc_exit(void) {
  remove_proc_entry("myxtime", NULL);
}

MODULE_LICENSE("GPL");
module_init(xtime_proc_init);
module_exit(xtime_proc_exit);
```

4.2 Dereferencing a null pointer

Next, I tried to dereference a null pointer. I did not get an immediate crash, but the logs showed an error.

```
[ 5297.159493] CPU: 0 PID: 4139 Comm: insmod Tainted: P           O 3.10.34 #6
[ 5297.159495] Hardware name: Hewlett-Packard HP Z210 Workstation/1588h, BIOS J51 v01
[ 5297.159498]  ffffffff81393ac2 0000000000000164 ffffffff81031a7c ffff880224c73340
[ 5297.159503]  ffff88021cf9bd08 0000000000000286 ffff880223890cc0 ffffffff81643860
[ 5297.159507]  0000000000000000 ffff88021d817740 ffffffff81031b32 ffffffff814efaf2
[ 5297.159512] Call Trace:
```

```
[ 5297.159518]  [<ffffffff81393ac2>] ? dump_stack+0x10/0x1e
[ 5297.159524]  [<ffffffff81031a7c>] ? warn_slowpath_common+0x5f/0x77
[ 5297.159528]  [<ffffffff81031b32>] ? warn_slowpath_fmt+0x45/0x4a
[ 5297.159533]  [<ffffffff81151120>] ? proc_alloc_inum+0x47/0xa0
[ 5297.159537]  [<ffffffff81151253>] ? proc_register+0xda/0x10b
[ 5297.159542]  [<ffffffff81151312>] ? proc_create_data+0x8e/0xa6
[ 5297.159546]  [<ffffffffa000e000>] ? 0xffffffffa000dfff
[ 5297.159551]  [<ffffffffa000e01e>] ? xtime_proc_init+0x1e/0x1000 [xtime_proc_nullpt
[ 5297.159555]  [<ffffffff81000240>] ? do_one_initcall+0x78/0x104
[ 5297.159560]  [<ffffffff8107ce21>] ? load_module+0x13e4/0x1690
[ 5297.159563]  [<ffffffff8107a94a>] ? set_section_ro_nx+0x58/0x58
[ 5297.159567]  [<ffffffff8107d203>] ? SyS_init_module+0xb9/0xbd
[ 5297.159572]  [<ffffffff8139b052>] ? system_call_fastpath+0x16/0x1b
[ 5297.159575] ---[ end trace e5f6b208aba6d633 ]---
```

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

static int xtime_proc_show(struct seq_file *m, void *v) {
  struct timespec now;
  getnstimeofday(&now);
  seq_printf(m, "%ld %ld\n", now.tv_sec, now.tv_nsec);
  return 0;
}

static int xtime_proc_open(struct inode *inode, struct  file *file) {
  return single_open(file, xtime_proc_show, NULL);
}

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

static int __init xtime_proc_init(void) {
  proc_create("myxtime", 0, NULL, &proc_fops);
  int* ptr = NULL;
  int ref = *ptr;
  return 0;
}

static void __exit xtime_proc_exit(void) {
  remove_proc_entry("myxtime", NULL);
}

MODULE_LICENSE("GPL");
module_init(xtime_proc_init);
module_exit(xtime_proc_exit);
```

4.3 Returning Non-zero from Initialization

Next, I returned a non-zero value from the `init` function. This had a similar effect as the previous attempt; however, the module seemed to just ignore the error, as seen in the following log:

```
[ 5869.234331] do_init_module: 'xtime_proc_nonzeroinit'->init suspiciously returned 5
[ 5869.234331] do_init_module: loading module anyway..
```

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

static int xtime_proc_show(struct seq_file *m, void *v) {
  struct timespec now;
  getnstimeofday(&now);
  seq_printf(m, "%ld %ld\n", now.tv_sec, now.tv_nsec);
  return 0;
}

static int xtime_proc_open(struct inode *inode, struct  file *file) {
  return single_open(file, xtime_proc_show, NULL);
}

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

static int __init xtime_proc_init(void) {
  proc_create("myxtime", 0, NULL, &proc_fops);
  //return 0;
}

static void __exit xtime_proc_exit(void) {
  remove_proc_entry("myxtime", NULL);
}

MODULE_LICENSE("GPL");
module_init(xtime_proc_init);
module_exit(xtime_proc_exit);
```

4.4 Calling the panic Function

Finally, wanting to truly see what a kernel panic would look like, without causing too much damage or spending too much more time, I called `panic()` from kernel-level code and shutdown the system.

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

static int xtime_proc_show(struct seq_file *m, void *v) {
  struct timespec now;
  getnstimeofday(&now);
  seq_printf(m, "%ld %ld\n", now.tv_sec, now.tv_nsec);
  return 0;
}

static int xtime_proc_open(struct inode *inode, struct  file *file) {
  return single_open(file, xtime_proc_show, NULL);
}
```

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

static int __init xtime_proc_init(void) {
  panic("Oh no!");
  proc_create("myxtime", 0, NULL, &proc_fops);
  return 0;
}

static void __exit xtime_proc_exit(void) {
  remove_proc_entry("myxtime", NULL);
}

MODULE_LICENSE("GPL");
module_init(xtime_proc_init);
module_exit(xtime_proc_exit);
```