CS 423
Operating System Design: Introduction to Linux Kernel Programming
(MP1 Walkthrough)

Andrew Yoo
(Some content taken from a previous year's walkthrough by Alberto Alvarez)
MP1 Goals

- Learn the basics of Linux kernel programming
- Learn the kernel implementation of linked lists
- Learn how to set up communication between the kernel and user space through procfs
- Also learn timers, interrupts, concurrency, etc.
Kernel Programming

- No memory protection
  - Driver, modules, and kernel threads all share same space
  - Don't crash your system!!
  - Don't corrupt your system!!
- Less reliable preemption
  - Deadlocks? CPU hogging? Concurrency = headache?
- Lack of user space libraries
- No floating point support
- No signals or security descriptors
• If your VMs fail, instructors are happy to help
• HOWEVER, you should try to avoid these problems
  • It can cost you valuable time
• Three ways:
  • Regularly snapshot your VM, but not too much
  • Push to your repository (basically no limit)
  • Keep track of your logs in /var
MP1 Overview

- Kernel module that measures CPU time of process
- Simple application that uses this service
- Proc filesystem to create a communication line between user space and kernel
  - `/proc/mp1/status`
- Two halves interrupt
  - Top half – Interrupt handler
  - Bottom half – Worker thread
Linux Kernel Module (LKM)

- LKM is code that is loaded and unloaded into the kernel on demand
- Not necessary to change kernel source code
- Entry and exit functions
- Compilation and runtime linkage different

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

int __init mpl_init(void){
    printk(KERN_ALERT "Hello, World\n");
    return 0;
}

void __exit mpl_exit(void){
    printk(KERN_ALERT "Goodbye, World\n");
}

module_init(myinit);
module_exit(myexit);
MODULE_LICENSE("GPL");
```
LKM “Hello World”

```c
#define LINUX
#include <linux/module.h>
#include <linux/kernel.h>
#include "mpl_given.h"

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Group ID");
MODULE_DESCRIPTION("CS-423 MP1");

#define DEBUG 1

// mpl_init - Called when module is loaded
int __init mpl_init(void)
{
    printk(KERN_ALERT "Hello, World\n");
    return 0;
}

// mpl_exit - Called when module is unloaded
void __exit mpl_exit(void)
{
    printk(KERN_ALERT "Goodbye, World\n");
}

// Register init and exit functions
module_init(mpl_init);
module_exit(mpl_exit);
```

- Edit source file as above
- Makefile is provided for MP1 (can be reused for MP2 and MP3)
LKM “Hello World”

abyoo2@sp20-cs423-005:~ /npl/demo$ make
rm -f userapp *.ko *.o *.mod.c Module.symvers modules.order
make -C /lib/modules/4.4.0-abyoo2/build M=/home/abyoo2/npl/demo modules
make[1]: Entering directory '/usr/src/linux-headers-4.4.0-abyoo2'
  CC [M] /home/abyoo2/npl/demo/npl.o
  Building modules, stage 2.
  MODPLAT 1 modules
  CC /home/abyoo2/npl/demo/npl.mod.o
  LD [M] /home/abyoo2/npl/demo/npl.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.4.0-abyoo2'
gcc -o userapp userapp.c
abyoo2@sp20-cs423-005:~ /npl/demo$ ls
Makefile modules.order Module.symvers npl.c npl_given.h npl.ko npl.mod.c npl.mod.o npl.o userapp userapp.c userapp.h
abyoo2@sp20-cs423-005:~ /npl/demo$
• sudo insmod hello.ko
  • "Installs" the module
• lsmod
  • Shows installed modules, including mp1
LKM “Hello World”

- modinfo
  - Lists the modules information
• `sudo rmmod hello`
  • Uninstalls the module
• `dmesg | tail -n`
  • `dmesg` checks kernel messages
  • `tail -n` prints the last n lines
  • Use these to debug
### Kernel Module (LKM)
- Starts with `module_init()`
- Runs in kernel space
- Does nothing until the kernel explicitly calls a module function
- Finishes with `module_exit()`

### Application
- Start with `main()`
- Runs in user space
- Executes through each lines
- Terminates
Functions available to LKM

- Applications have access to library functions
  - `printf()`, `malloc()`, `free()`
- Kernel modules need to use library functions provided by kernel:
  - `printk()`, `kmalloc()`, `kfree()`, `vmalloc()`
  - `/proc/kallsyms` lists kernel provided functions
- Linux Kernel Programming Guide page and references on the MP1 page
The /proc file system

- Virtual file system
- Allows communication between kernel and user space
- Does not contain 'real' files
- Contains runtime system information
  - System memory, hardware configuration, etc.

The /proc file system

```
abyoo2@sp29-CS423-80S:/proc$ cat cpuinfo
processor : 0
vendor_id : GenuineIntel
cpu family : 6
model : 63
model name : Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz
stepping : 2
microcode : 0x43
cpu MHz : 2299.990
cache size : 30720 KB
physical id : 0
siblings : 1
core id : 0
cpu cores : 1
apicid : 0
initial apicid : 0
fpu : yes
fpu_exception : yes
cpuid level : 15
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep ntrr pge mca cmov pat
```
Using /proc in MP1

• Create a directory using proc_mkdir()
  • Arguments: name and parent (proc_dir_entry*)
  • Returns proc_dir_entry*

• Create a file using proc_create
  • Arguments: name, mode (permissions), parent, pointer to file operations
  • returns proc_dir_entry*
struct file_operations {
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
    ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
    ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
    ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
    ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
    int (*iterate) (struct file *, struct dir_context *);
    unsigned int (*poll) (struct file *, struct poll_table_struct *);
    long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
    long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
    int (* mmap) (struct file *, struct vm_area_struct *);
    int (* open) (struct inode *, struct file *);
    int (* flush) (struct file *, fl_owner_t id);
    int (* release) (struct inode *, struct file *);
    int (* fsync) (struct file *, loff_t, loff_t, int datasync);
    int (* aio fsync) (struct kiocb *, int datasync);
    int (* fasync) (int, struct file *, int);
    int (* lock) (struct file *, int, struct file_lock *);
    ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
    unsigned long (* get_unmapped_area) (struct file *, unsigned long, unsigned long, unsigned long, unsigned long, unsigned long);
    int (* check_flags) (int);
    int (* flock) (struct file *, int, struct file_lock *);
    ssize_t (* splice_write) (struct pipe_inode_info *, struct file *, loff_t *, size_t, unsigned int);
    ssize_t (* splice_read) (struct file *, loff_t *, struct pipe_inode_info *, size_t, unsigned int);
    int (* setlease) (struct file *, long, struct file_lock **, void **);
    long (* fallocate) (struct file *, file, int mode, loff_t offset, loff_t len);
    int (* show_fdinfo) (struct seq_file *, struct file *);
};
Using /proc in MP1

Sample code:

```c
#define FILENAME "status"
#define DIRECTORY "mpl"
static struct proc_dir_entry *proc_dir;
static struct proc_dir_entry *proc_entry;
static ssize_t mp1_read (struct file *file, char __user *buffer, size_t count, loff_t *data)
    // implementation goes here...
}
static ssize_t mp1_write (struct file *file, const char __user *buffer, size_t count, loff_t *data)
    // implementation goes here...
}
static const struct file_operations mp1_file = {
    .owner = THIS_MODULE,
    .read  = mp1_read,
    .write = mp1_write,
};
int __init mp1_init(void)
{
    proc_dir = proc_mkdir(DIRECTORY, NULL);
    proc_entry = proc_create(FILENAME, 0666, proc_dir, & mp1_file);
}
```
Using /proc in MP1

- Within MP1_read/mp1_write, you may need to move data between kernel/user space
  - copy_from_user()
  - copy_to_user()

Sample code (There are other ways of implementing it):

```c
static ssize_t mp1_read (struct file *file, char __user *buffer, size_t count, loff_t *data){
    // implementation goes here...
    int copied;
    char * buf;
    buf = (char *) kmalloc(count,GFP_KERNEL);
    copied = 0;
    //... put something into the buf, updated copied
    copy_to_user(buffer, buf, copied);
    kfree(buf);
    return copied;
}
```
You will use Linux list to store all registered user processes
Linux kernel list is a widely used data structure in Linux kernel
• Defined in `<linux/linux.h>`
• You MUST get familiar of how to use it

```c
struct list_head{
    struct list_head *next;
    struct list_head *prev;
};

struct my_cool_list{
    struct list_head list; /* kernel's list structure */
    int my_cool_data;
    void* my_cool_void;
};
```
Linux Kernel Lists

- Some useful API calls:
  - `LIST_HEAD(new_list)`
  - `list_add(struct list_head *new, struct list_head *head)`
  - `list_for_each_safe(pos, n, head)`
  - `list_entry(ptr, type, member)`
  - `list_del(pos)`
  - `list_for_each_entry(pos, head, member)`
  - `list_empty(ptr)`
Kernel Timer

- Operate in units called `jiffies', not seconds
  - `msec_to_jiffies()` converts ms to jiffies
  - `jiffies_to_msec()` converts jiffies to ms

```c
struct timer_list {
    /* ... */
    unsigned long expires;
    void (*function)(unsigned long);
    unsigned long data;
};
```
• Some useful API calls:

```c
void setup_timer(struct timer_list *timer, 
    void(*function)(unsigned long), unsigned long data)

int mod_timer(struct timer_list *timer, unsigned long expires)

void del_timer(struct timer_list *timer)

void init_timer(struct timer_list *timer);

struct timer_list TIMER_INITIALIZER(_function, _expires, _data);

void add_timer(struct timer_list * timer);
```
Work queues

• Request a function to be called at some time
  • Workqueue functions can sleep
  • Can be used to implement bottom half

• Some useful API calls:

  INIT_WORK (struct work_struct *work, void (*function) (void *), void *data)

  void flush_workqueue (struct workqueue_struct *queue)

  void destroy_workqueue (struct workqueue_struct *queue)

  int queue_work (struct workqueue_struct *queue, struct work_struct *work)
Don’t forget about Office hours & Piazza!