



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



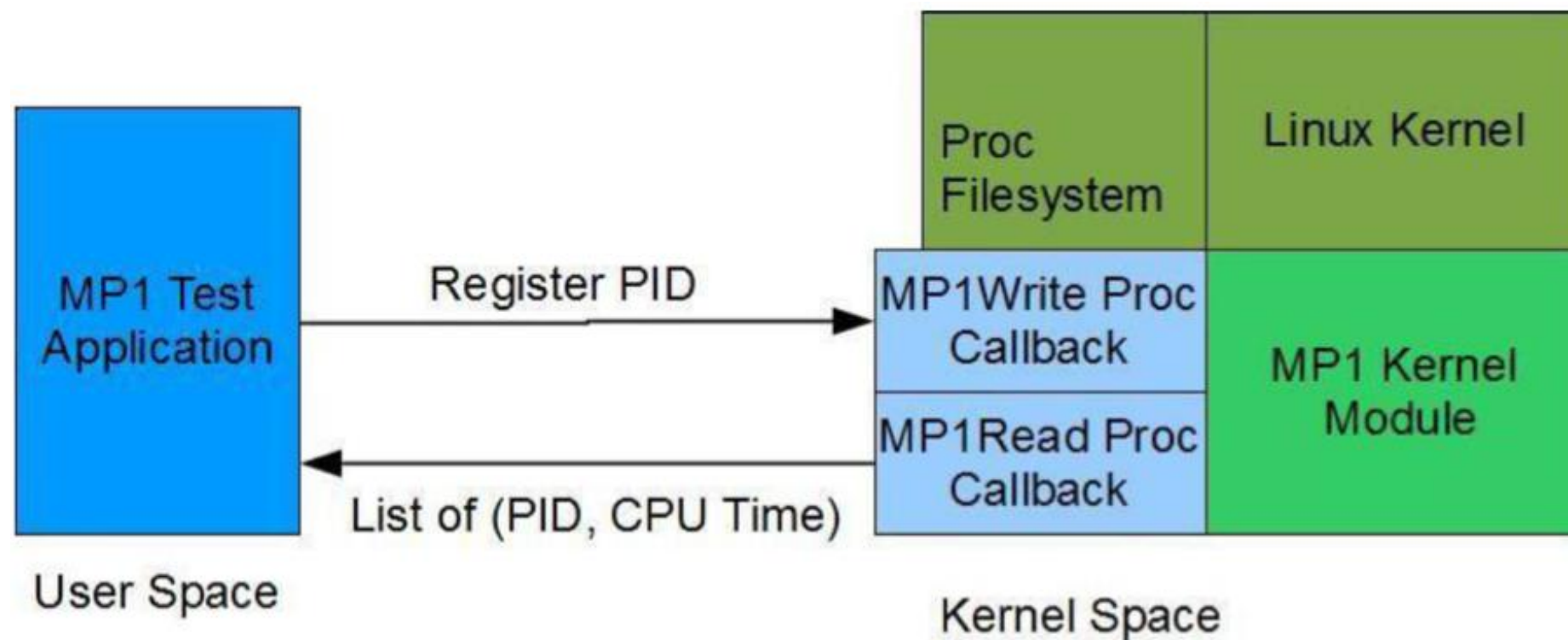
- No memory protection
 - Driver, modules, and kernel threads all share same space
 - Don't crash your system!!
 - Don't corrupt your system!!
- Less reliable preemption
 - Deadlocks? CPU hogging? Concurrency = headache?
- Lack of user space libraries
- No floating point support
- No signals or security descriptors

Be Careful



- 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

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

int __init mp1_init(void){
    printk(KERN_ALERT "Hello, World\n");
    return 0;
}

void __exit mp1_exit(void){
    printk(KERN_ALERT "Goodbye, World\n");
}

module_init(myinit);
module_exit(myexit);
MODULE_LICENSE("GPL");
```

LKM “Hello World”



```
#define LINUX

#include <linux/module.h>
#include <linux/kernel.h>
#include "mp1_given.h"

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

#define DEBUG 1

// mp1_init - Called when module is loaded
int __init mp1_init(void)
{
    printk(KERN_ALERT "Hello, World\n");
    return 0;
}

// mp1_exit - Called when module is unloaded
void __exit mp1_exit(void)
{
    printk(KERN_ALERT "Goodbye, World\n");
}

// Register init and exit functions
module_init(mp1_init);
module_exit(mp1_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:~/mp1/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/mp1/demo modules
make[1]: Entering directory '/usr/src/linux-headers-4.4.0-abyoo2'
  CC [M] /home/abyoo2/mp1/demo/mp1.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/abyoo2/mp1/demo/mp1.mod.o
  LD [M] /home/abyoo2/mp1/demo/mp1.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.4.0-abyoo2'
gcc -o userapp userapp.c
abyoo2@sp20-cs423-005:~/mp1/demo$ ls
Makefile modules.order Module.symvers mp1.c mp1_given.h mp1.ko mp1.mod.c mp1.mod.o mp1.o userapp userapp.c userapp.h
abyoo2@sp20-cs423-005:~/mp1/demo$
```


LKM “Hello World”



```
abyoo2@sp20-cs423-005:~/mp1/demo$ ls
Makefile  modules.order  Module.symvers  mp1.c  mp1_given.h  mp1.ko  mp1.mod.c  mp1.mod.o  mp1.o  userapp  userapp.c  userapp.h
abyoo2@sp20-cs423-005:~/mp1/demo$ sudo insmod mp1.ko
[sudo] password for abyoo2:
abyoo2@sp20-cs423-005:~/mp1/demo$ lsmod
Module                  Size  Used by
mp1 08x223 (0.2MP)      1259  0
```

- `sudo insmod hello.ko`
 - "Installs" the module
- `lsmod`
 - Shows installed modules, including `mp1`

LKM “Hello World”



```
abyoo2@sp20-cs423-005: ~/mp1/demo$ modinfo mp1.ko
filename:          /home/abyoo2/mp1/demo/mp1.ko
description:       CS-423 MP1
author:            Group_ID
license:           GPL
srcversion:        CFC9C46D984AA03A4699A82
depends:
vermagic:          4.4.0-abyoo2 SMP mod_unload
abyoo2@sp20-cs423-005: ~/mp1/demo$
```

- modinfo
 - Lists the modules information

LKM “Hello World”



```
abyoo2@sp20-cs423-005: ~/mp1/demo$ sudo rmmod mp1
abyoo2@sp20-cs423-005: ~/mp1/demo$ lsmod
Module                Size  Used by
st                    49357   0
lp                    10271   0
```

- `sudo rmmod hello`
 - Uninstalls the module

LKM “Hello World”



```
abyoo2@sp20-cs423-005: ~/mp1/demo$ dmesg | tail -2  
[595856.856798] Hello, World  
[596179.876272] Goodbye, World  
abyoo2@sp20-cs423-005: ~/mp1/demo$
```

- dmesg | tail -n
 - dmesg checks kernel messages
 - tail -n prints the last n lines
 - Use these to debug

Kernel vs. Application Programming



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.

<http://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html>

The /proc file system



```
abyoo2@sp20-cs423-005:/proc$ ls
1      1038  1206  1346  13782 1465  15159 205  215  224  236  246  257  266  2880 31      39  461  478  616  813  848  861  911  acpi  dma      kallsyms  mdstat  sched_debug  sysvipc
10     107   1209  1350  13821 14651 15161 206  216  225  237  247  258  267  2881 32      392  462  546  617  815  849  862  92  buddyinfo driver  kcore     meminfo    schedstat   thread-self
100    1092  121   13566 13822 14661 16      207  217  227  238  248  259  268  29  32027 417  47  558  7    816  850  87  93  bus     execdomains keys     misc       scsi        timer_list
1009   11    122   1357  1409  14664 17      208  218  228  239  249  26  269  3  322  419  471  559  8    819  854  874  94  cgroups fb      key-users modules    self        tty
1012   11624 1239  1358  14314 14808 18      209  219  23  240  250  260  27  30  33  430  472  571  807  821  855  88  95  cmdline filesystems kmsg      kpagecgroup mpt         slabinfo    uptime
1017   1199  1252  13593 1444  15147 182     21  22  230  241  251  261  28  300 34  431  473  592  808  831  856  89  96  consoles fs      kpagecount mtr         softirqs    version
1019   12    13    13769 1454  15148 183     210  220  231  242  252  262  2876 301 35  44  474  594  809  844  857  9  97  cpuinfo interrupts kpagecount net         stat         vmallocinfo
1026   1201  1321  13773 1455  15150 184     212  221  233  243  253  263  2877 302 36  440  475  595  810  845  858  90  98  crypto  iomem   kpageflags loadavg     swaps        vmstat
1032   1202  1337  13776 1456  15152 2      213  222  234  244  254  264  2878 304 37  45  476  596  811  846  859  907 9891 devices ioports  locks      pagetypeinfo sys          zoneinfo
1036   1205  1338  13778 1463  15156 20      214  223  235  245  256  265  2879 306 38  46  477  614  812  847  860  91  99  diskstats irq      locks      partitions  sysrq-trigger
```


The /proc file system



```
abyoo2@sp20-cs423-005:/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       : 2299.998
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 mtrr 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*`

Using /proc in MP1



```
1486 struct file_operations {
1487     struct module *owner;
1488     loff_t (*llseek) (struct file *, loff_t, int);
1489     ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
1490     ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
1491     ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
1492     ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
1493     ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
1494     ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
1495     int (*iterate) (struct file *, struct dir_context *);
1496     unsigned int (*poll) (struct file *, struct poll_table_struct *);
1497     long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
1498     long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
1499     int (*mmap) (struct file *, struct vm_area_struct *);
1500     int (*open) (struct inode *, struct file *);
1501     int (*flush) (struct file *, fl_owner_t id);
1502     int (*release) (struct inode *, struct file *);
1503     int (*fsync) (struct file *, loff_t, loff_t, int datasync);
1504     int (*aio_fsync) (struct kiocb *, int datasync);
1505     int (*fasync) (int, struct file *, int);
1506     int (*lock) (struct file *, int, struct file_lock *);
1507     ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
1508     unsigned long (*get_unmapped_area) (struct file *, unsigned long, unsigned long, unsigned long, unsigned
1509     int (*check_flags) (int);
1510     int (*flock) (struct file *, int, struct file_lock *);
1511     ssize_t (*splice_write) (struct pipe_inode_info *, struct file *, loff_t *, size_t, unsigned int);
1512     ssize_t (*splice_read) (struct file *, loff_t *, struct pipe_inode_info *, size_t, unsigned int);
1513     int (*setlease) (struct file *, long, struct file_lock **, void **);
1514     long (*fallocate) (struct file *file, int mode, loff_t offset,
1515                        loff_t len);
1516     int (*show_fdinfo) (struct seq_file *m, struct file *f);
1517 };
```

Using /proc in MP1



Sample code:

```
#define FILENAME "status"
#define DIRECTORY "mp1"
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):

```
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 ;
}
```

Linux Kernel Lists



- 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

```
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

```
struct timer_list {  
    /* ... */  
    unsigned long expires;  
    void (*function)(unsigned long);  
    unsigned long data;  
};
```


Kernel Timer



- **Some useful API calls:**

```
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)
```

Questions??



Don't forget about Office hours & Piazza!