Introduction to Linux Kernel Programming

CS423 2016 Spring
Haitong Tian

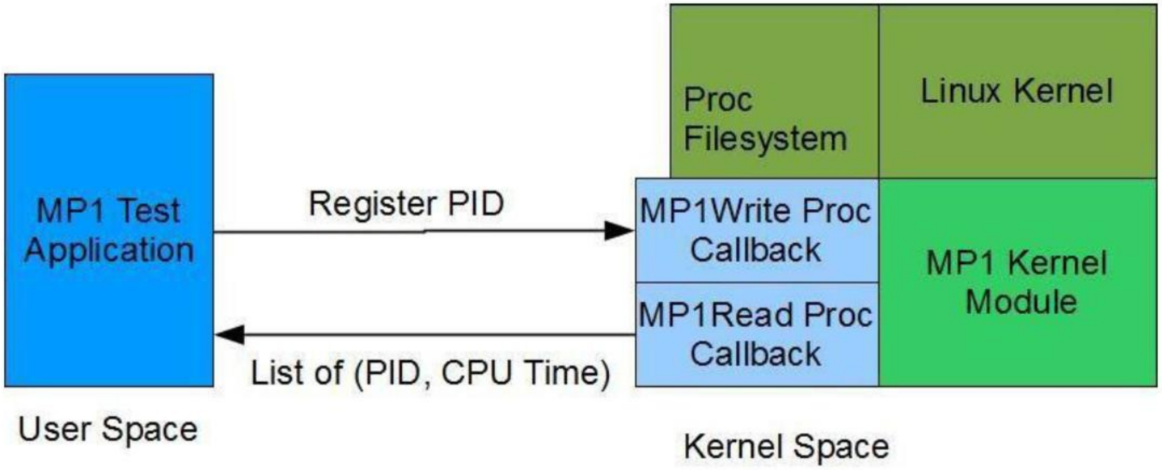
ILLINOIS

illinois.edu

Purpose of MP1

- Get yourself familiar with Linux kernel programming
- Learn to use kernel linked list data structure
- Learn to use /proc to communicate between kernel and use space program
- Timer, workqueue, interrupt etc.

MP1 Overview



- Build kernel module measure user app cpu time
- Use /proc file system to communicate between user program and kernel module
  - /proc/mp1/status
- Two-halves interrupt handler implementation
  - Top-half: interrupt handler
  - Bottom half: workqueue + worker thread

Kernel vs Application Programming

Kernel

- No memory protection
  - Share memory with devices, scheduler
  - Easily crash the system, very hard to debug
- Sometimes no preemption
  - Can hog the CPU
  - Concurrency is hard
- No libraries
  - No printf, fopen
- No access to files
- Direct access to hardware

Application

- Memory protection
  - Segmentation fault
  - Can conveniently debug the program
- Preemption
  - Scheduling is not our responsibility
- Signals (Control-C)
- Libraries
- In Linux, everything is a file descriptor
- Access to hardware as files

Linux Kernel Module (LKM)

- LKM are pieces of code that can be loaded and unloaded into the kernel upon demand
  - No need to modify the kernel source code for all MPs in this course
- Separate compilation
- Runtime linkage
- Entry and Exit functions

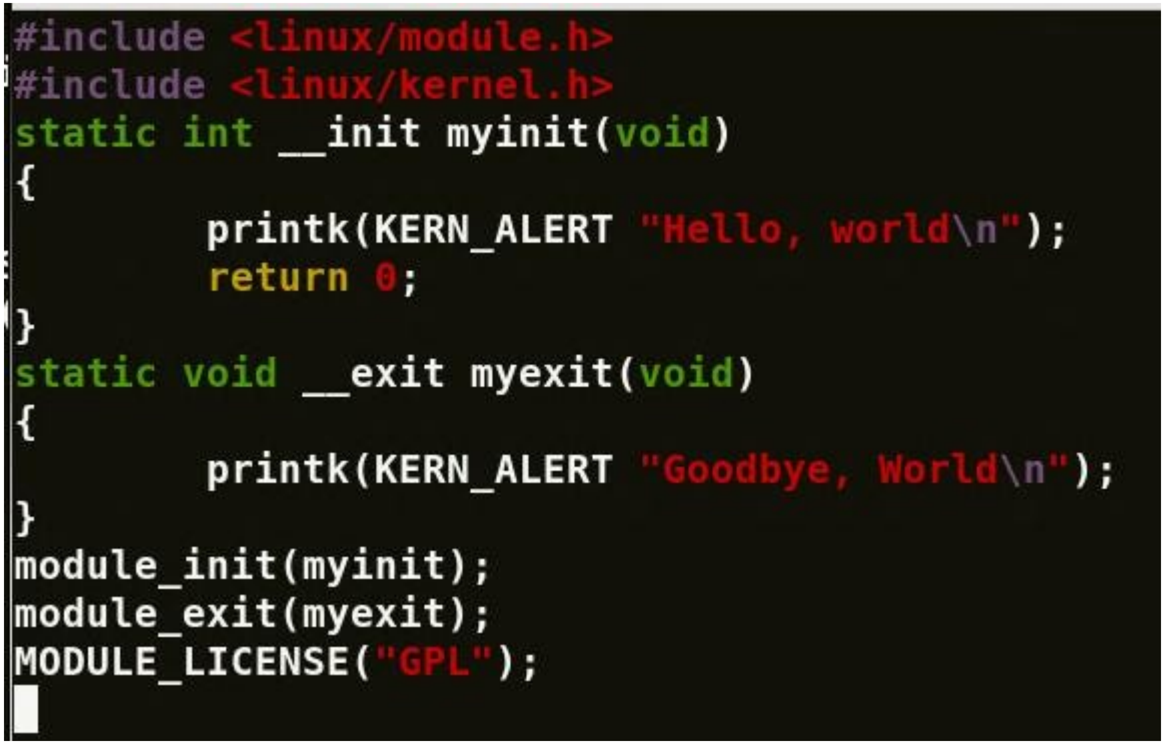
```
#include <linux/module.h>
#include <linux/kernel.h>

static int __init myinit(void)
        {  printk(KERN_ALERT "Hello, world
        \n");  return 0;
}

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

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

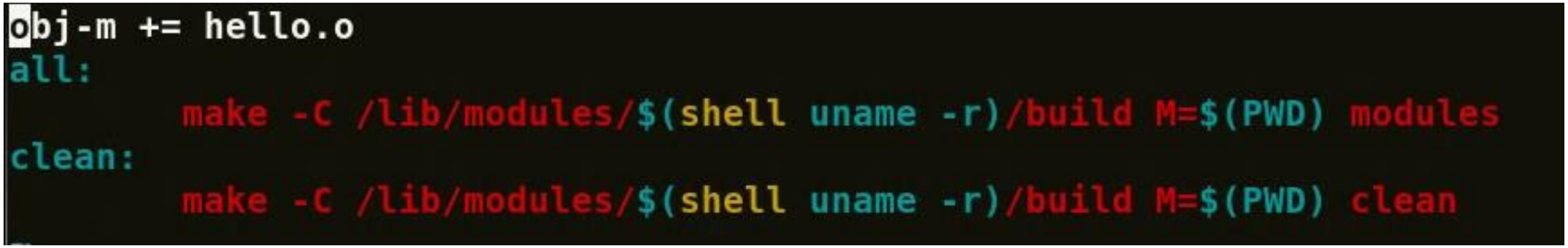
LKM “Hello World”



```
#include <linux/module.h>
#include <linux/kernel.h>
static int __init myinit(void)
{
        printk(KERN_ALERT "Hello, world\n");
        return 0;
}
static void __exit myexit(void)
{
        printk(KERN_ALERT "Goodbye, World\n");
}
module_init(myinit);
module_exit(myexit);
MODULE_LICENSE("GPL");
```

- Edit source file as above

LKM “Hello World”

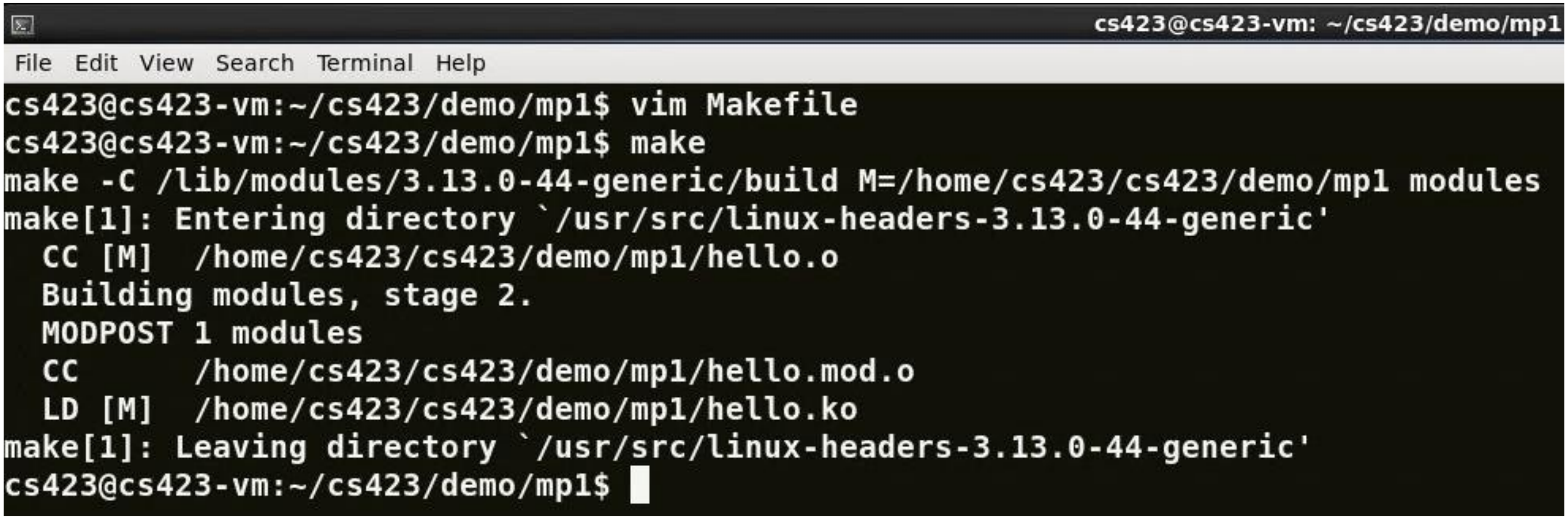


```
obj-m += hello.o
all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

- Edit the Makefile

- For MP1, the Makefile is provided
  - It can be reused for MP2/MP3

LKM “Hello World”

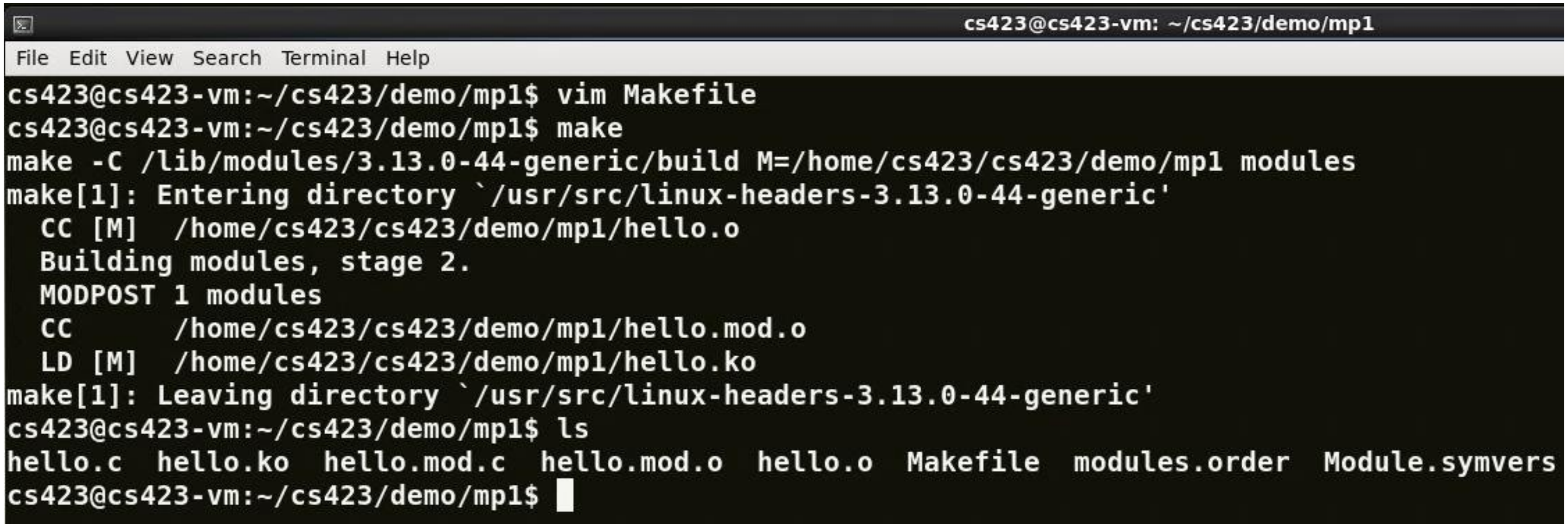


cs423@cs423-vm: ~/cs423/demo/mp1
File Edit View Search Terminal Help

```
cs423@cs423-vm:~/cs423/demo/mp1$ vim Makefile
cs423@cs423-vm:~/cs423/demo/mp1$ make
make -C /lib/modules/3.13.0-44-generic/build M=/home/cs423/cs423/demo/mp1 modules
make[1]: Entering directory `/usr/src/linux-headers-3.13.0-44-generic'
  CC [M]  /home/cs423/cs423/demo/mp1/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/cs423/cs423/demo/mp1/hello.mod.o
  LD [M]  /home/cs423/cs423/demo/mp1/hello.ko
make[1]: Leaving directory `/usr/src/linux-headers-3.13.0-44-generic'
cs423@cs423-vm:~/cs423/demo/mp1$ 
```

- Make
  - Compile the module

LKM “Hello World”

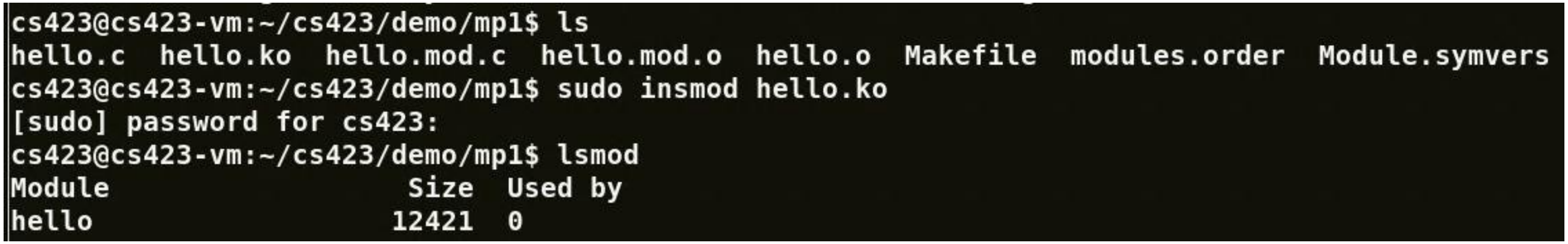


cs423@cs423-vm: ~/cs423/demo/mp1
File Edit View Search Terminal Help

```
cs423@cs423-vm:~/cs423/demo/mp1$ vim Makefile
cs423@cs423-vm:~/cs423/demo/mp1$ make
make -C /lib/modules/3.13.0-44-generic/build M=/home/cs423/cs423/demo/mp1 modules
make[1]: Entering directory `/usr/src/linux-headers-3.13.0-44-generic'
  CC [M]  /home/cs423/cs423/demo/mp1/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/cs423/cs423/demo/mp1/hello.mod.o
  LD [M]  /home/cs423/cs423/demo/mp1/hello.ko
make[1]: Leaving directory `/usr/src/linux-headers-3.13.0-44-generic'
cs423@cs423-vm:~/cs423/demo/mp1$ ls
hello.c  hello.ko  hello.mod.c  hello.mod.o  hello.o  Makefile  modules.order  Module.symvers
cs423@cs423-vm:~/cs423/demo/mp1$ 
```

- ls
  - Module is compiled as hello.ko

LKM “Hello World”

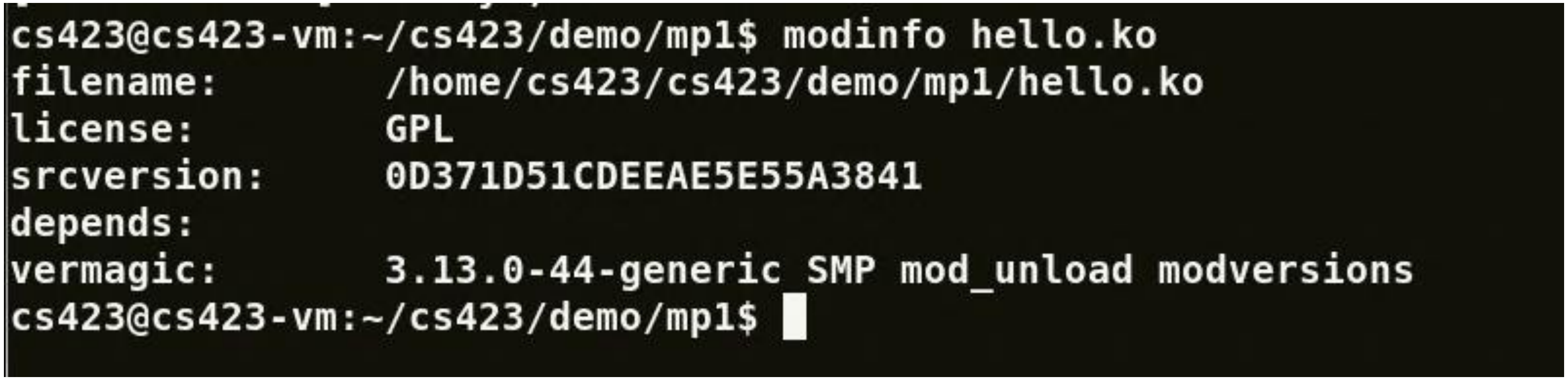


```
cs423@cs423-vm:~/cs423/demo/mp1$ ls
hello.c  hello.ko  hello.mod.c  hello.mod.o  hello.o  Makefile  modules.order  Module.symvers
cs423@cs423-vm:~/cs423/demo/mp1$ sudo insmod hello.ko
[sudo] password for cs423:
cs423@cs423-vm:~/cs423/demo/mp1$ lsmod
Module                  Size  Used by
hello                  12421  0
```

- sudo insmod hello.ko
  - install the module

- lsmod
  - You can see the hello module on the top of the list

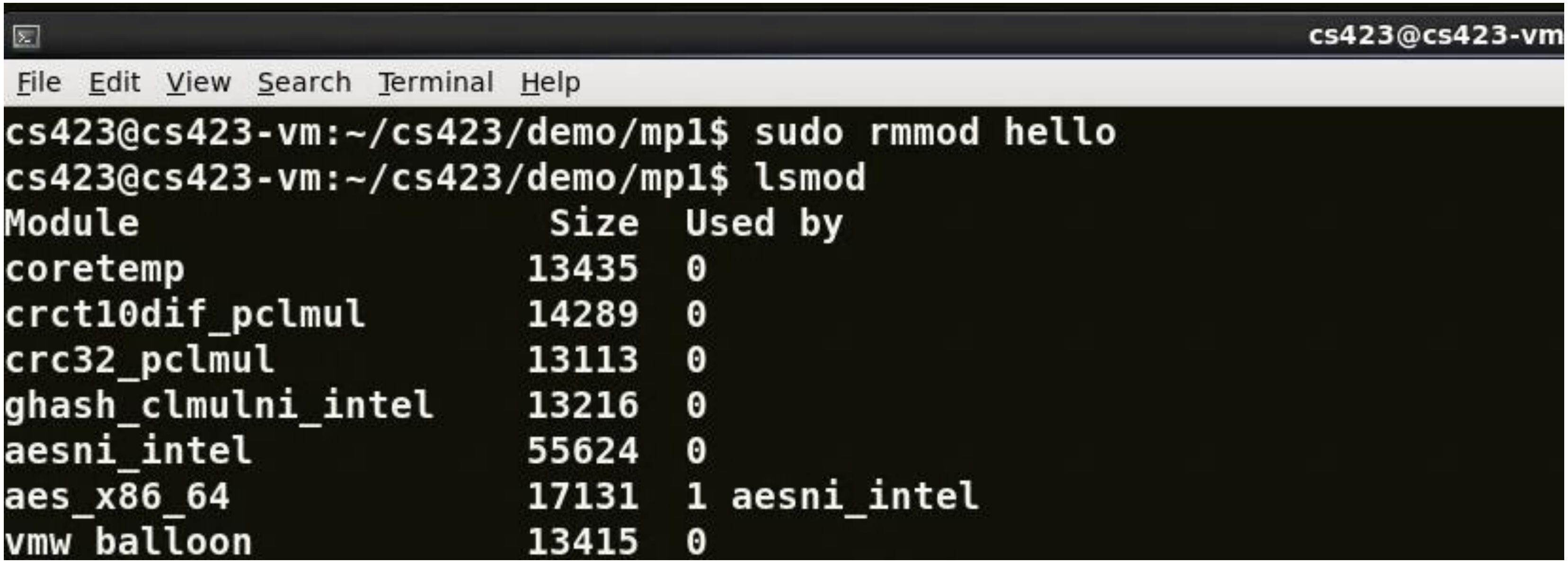
LKM “Hello World”



```
cs423@cs423-vm:~/cs423/demo/mp1$ modinfo hello.ko
filename:       /home/cs423/cs423/demo/mp1/hello.ko
license:        GPL
srcversion:     0D371D51CDEEAE5E55A3841
depends:
vermagic:       3.13.0-44-generic SMP mod_unload modversions
cs423@cs423-vm:~/cs423/demo/mp1$
```

- modinfo
  - Check the module information

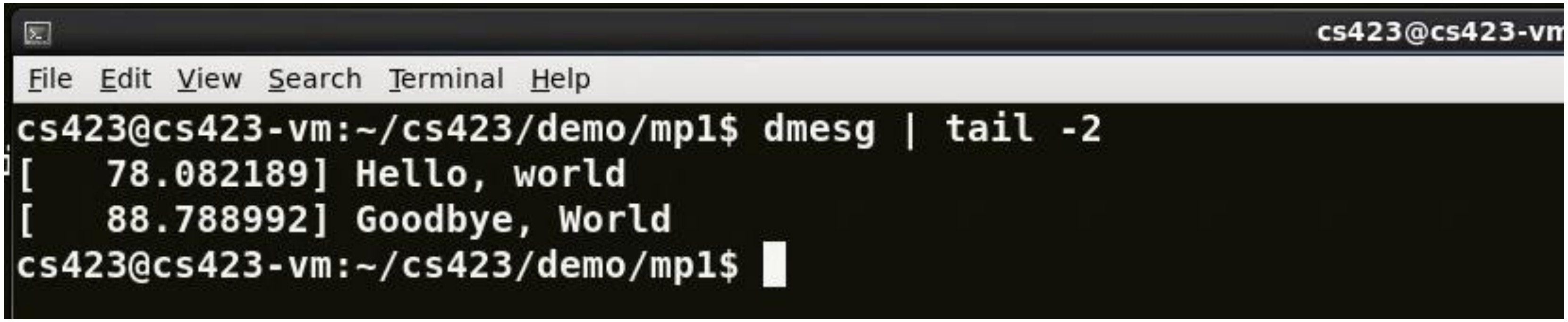
LKM “Hello World”



```
cs423@cs423-vm:~/cs423/demo/mp1$ sudo rmmod hello
cs423@cs423-vm:~/cs423/demo/mp1$ lsmod
Module                  Size  Used by
coretemp               13435  0
crct10dif_pclmul       14289  0
crc32_pclmul           13113  0
ghash_clmulni_intel    13216  0
aesni_intel            55624  0
aes_x86_64             17131  1 aesni_intel
vmw_balloon            13415  0
```

- sudo rmmod hello
  - uninstall the module

LKM “Hello World”



```
cs423@cs423-vm:~/cs423/demo/mp1$ dmesg | tail -2
[   78.082189] Hello, world
[   88.788992] Goodbye, World
cs423@cs423-vm:~/cs423/demo/mp1$ 
```

- dmesg
  - Check kernel messages (printk)
  - Very useful to debug the module
  - dmesg | tail –n
    - Check the last n lines of kernel message

LKM “Hello World”

- To summarize
  - sudo insmod hello.ko
    - install the kernel module
  - lsmod
    - Check if the module is loaded
    - All loaded modules can be found /proc/modules
  - sudo rmmod hello
    - Unload the module

Kernel Module vs Application Programming

Kernel Module (LKM)

- Start with module_init()
  - Set up the kernel
- Runs in kernel space
- The module does nothing until one of the module functions are called by the kernel
- Ends with module_exit()

Applications

- Start with main()
- Runs in user space
- Executes a bunch of instructions
- Terminates

Functions available to LKM

- Applications have access to library functions
  - printf(), malloc(), free()

- Kernel modules do not have access to library functions except that provided by kernel
  - printk(), kmalloc(), kfree(), vmalloc()
  - Check /proc/kallsyms to see a list of kernel provided functions

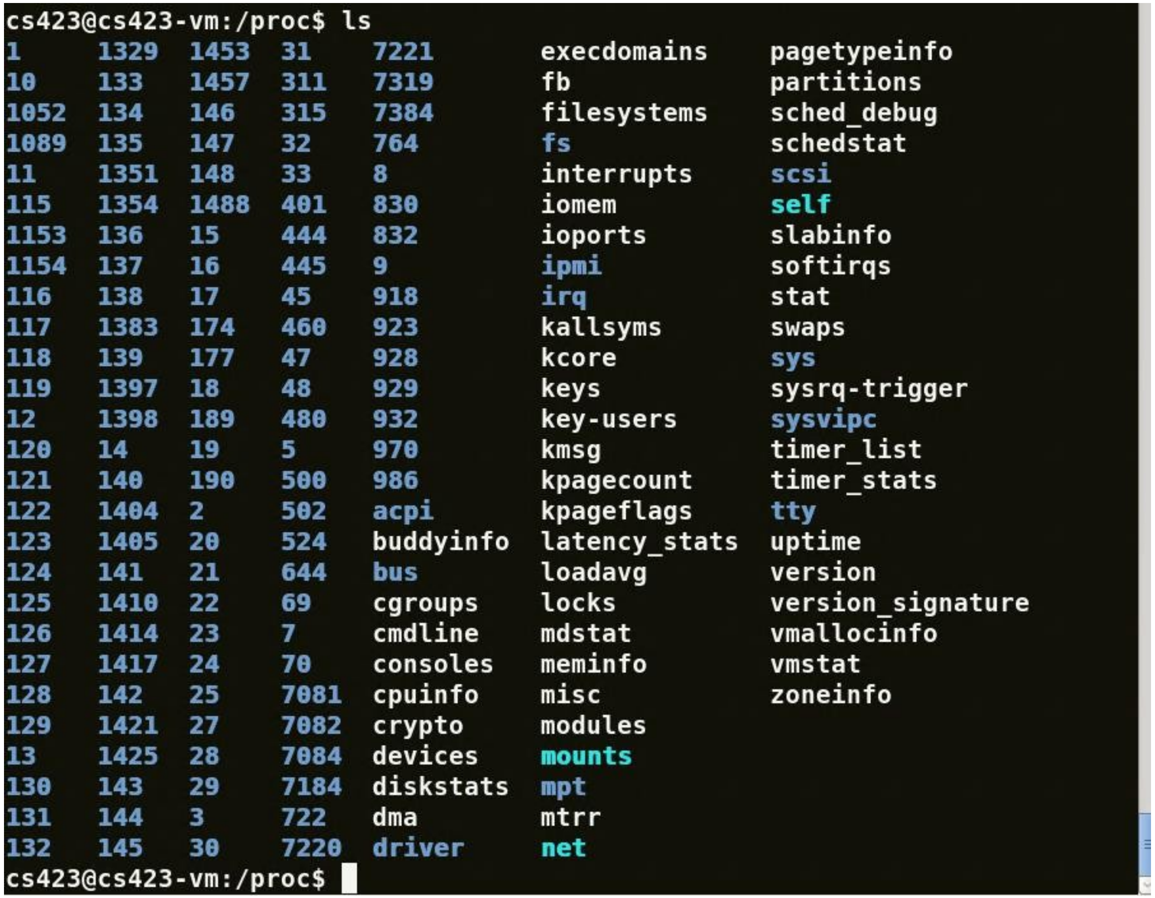
- Check Linux Kernel Programming Guide page and references on the MP1 page

The /proc file system

- /proc is a virtual file system that allow communication between kernel and use space
- It doesn't contain 'real' files but runtime system information
  - system memory, devices mounted, hardware configuration
- Widely used for many reportings
  - /proc/modules, /proc/meminfo, /proc/cpuinfo

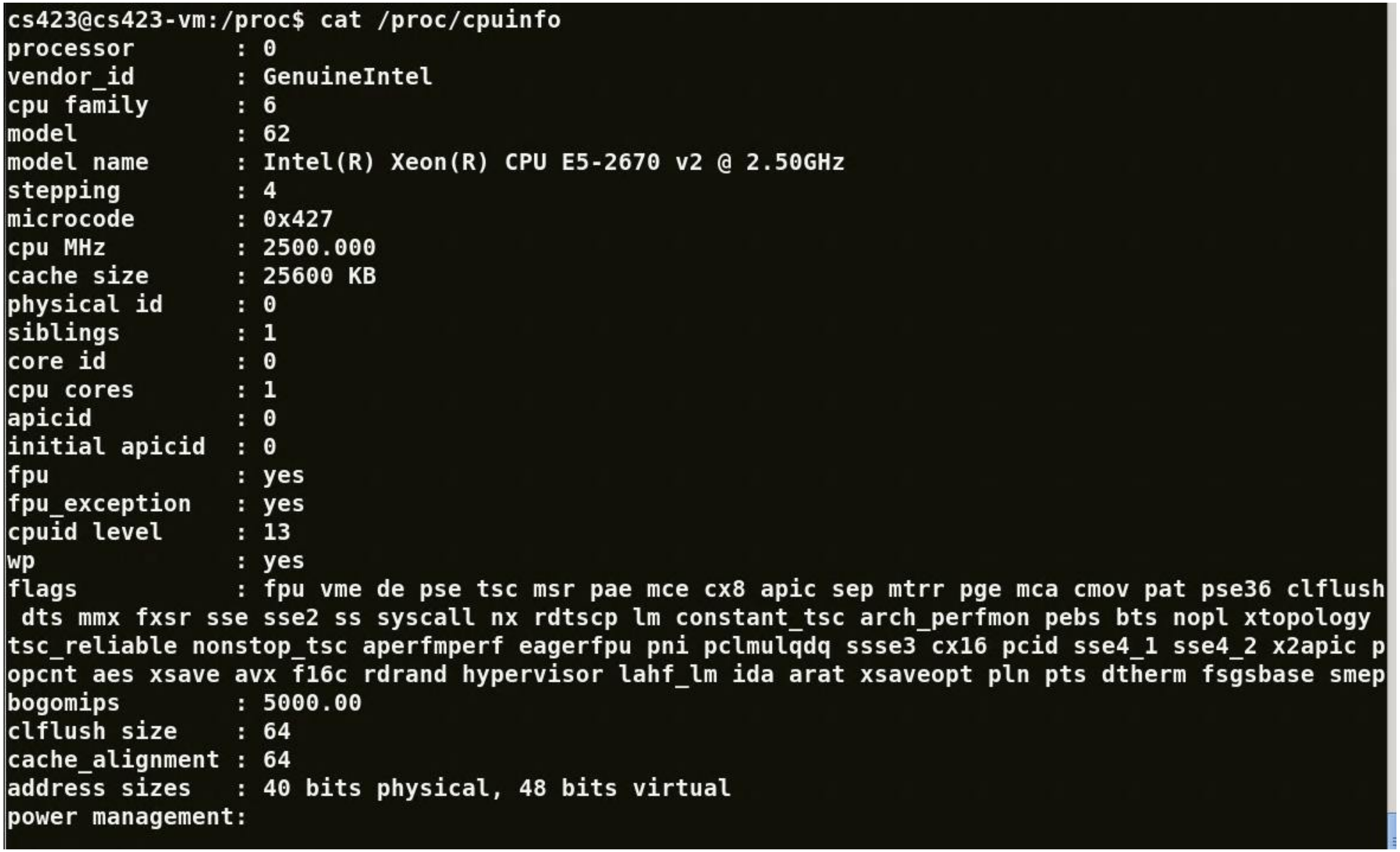
http://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html

The /proc file system



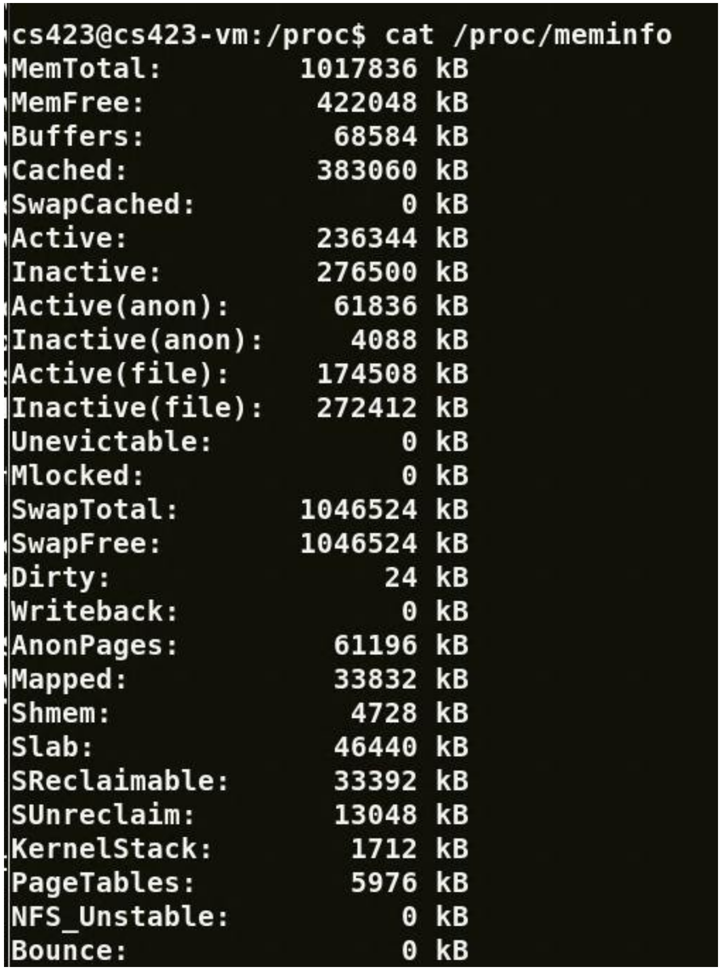
```
cs423@cs423-vm:/proc$ ls
1     1329  1453  31    7221       execdomains    pagetypeinfo
10    133   1457  311   7319       fb             partitions
1052  134   146   315   7384       filesystems    sched_debug
1089  135   147   32    764        fs             schedstat
11    1351  148   33    8          interrupts     scsi
115   1354  1488  401   830        iomem          self
1153  136   15    444   832        ioports        slabinfo
1154  137   16    445   9          ipmi           softirqs
116   138   17    45    918        irq            stat
117   1383  174   460   923        kallsyms       swaps
118   139   177   47    928        kcore          sys
119   1397  18    48    929        keys           sysrq-trigger
12    1398  189   480   932        key-users      sysvipc
120   14    19    5     970        kmsg           timer_list
121   140   190   500   986        kpagecount     timer_stats
122   1404  2     502   acpi       kpageflags     tty
123   1405  20    524   buddyinfo  latency_stats  uptime
124   141   21    644   bus        loadavg        version
125   1410  22    69    cgroups    locks          version_signature
126   1414  23    7     cmdline    mdstat         vmallocinfo
127   1417  24    70    consoles   meminfo        vmstat
128   142   25    7081  cpuinfo    misc           zoneinfo
129   1421  27    7082  crypto     modules
13    1425  28    7084  devices    mounts
130   143   29    7184  diskstats  mpt
131   144   3     722   dma        mtrr
132   145   30    7220  driver     net
cs423@cs423-vm:/proc$ 
```

The /proc file system



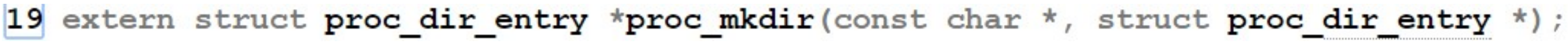
```
cs423@cs423-vm:/proc$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 62
model name      : Intel(R) Xeon(R) CPU E5-2670 v2 @ 2.50GHz
stepping        : 4
microcode       : 0x427
cpu MHz         : 2500.000
cache size      : 25600 KB
physical id     : 0
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush
 dts mmx fxsr sse sse2 ss syscall nx rdtscp lm constant_tsc arch_perfmon pebs bts nopl xtopology
tsc_reliable nonstop_tsc aperfmperf eagerfpu pni pclmulqdq ssse3 cx16 pcid sse4_1 sse4_2 x2apic p
opcnt aes xsave avx f16c rdrand hypervisor lahf_lm ida arat xsaveopt pln pts dtherm fsgsbase smep
bogomips        : 5000.00
clflush size    : 64
cache_alignment : 64
address sizes   : 40 bits physical, 48 bits virtual
power management:
```

The /proc file system



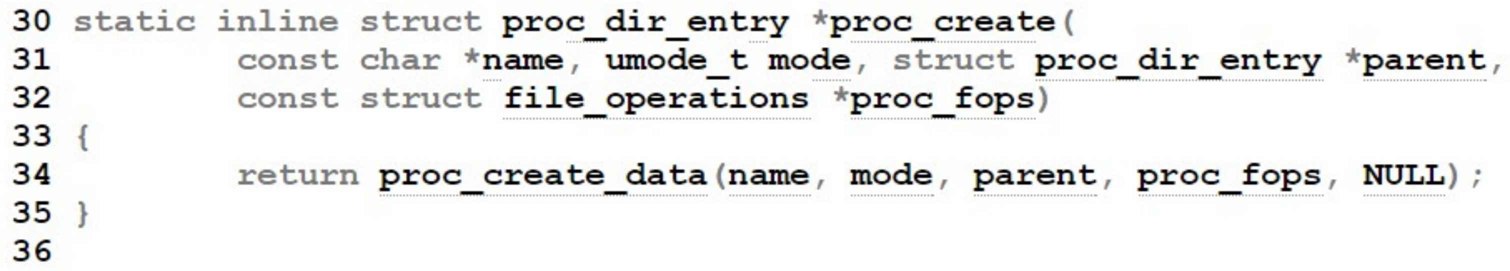
```
cs423@cs423-vm:/proc$ cat /proc/meminfo
MemTotal:        1017836 kB
MemFree:          422048 kB
Buffers:           68584 kB
Cached:           383060 kB
SwapCached:            0 kB
Active:           236344 kB
Inactive:         276500 kB
Active(anon):      61836 kB
Inactive(anon):     4088 kB
Active(file):     174508 kB
Inactive(file):   272412 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:       1046524 kB
SwapFree:        1046524 kB
Dirty:                24 kB
Writeback:             0 kB
AnonPages:         61196 kB
Mapped:            33832 kB
Shmem:              4728 kB
Slab:              46440 kB
SReclaimable:      33392 kB
SUnreclaim:        13048 kB
KernelStack:        1712 kB
PageTables:         5976 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
```

Using /proc in MP1



```
extern struct proc_dir_entry *proc_mkdir(const char *, struct proc_dir_entry *);
```

- Create a directory under /proc
  - proc_mkdir()

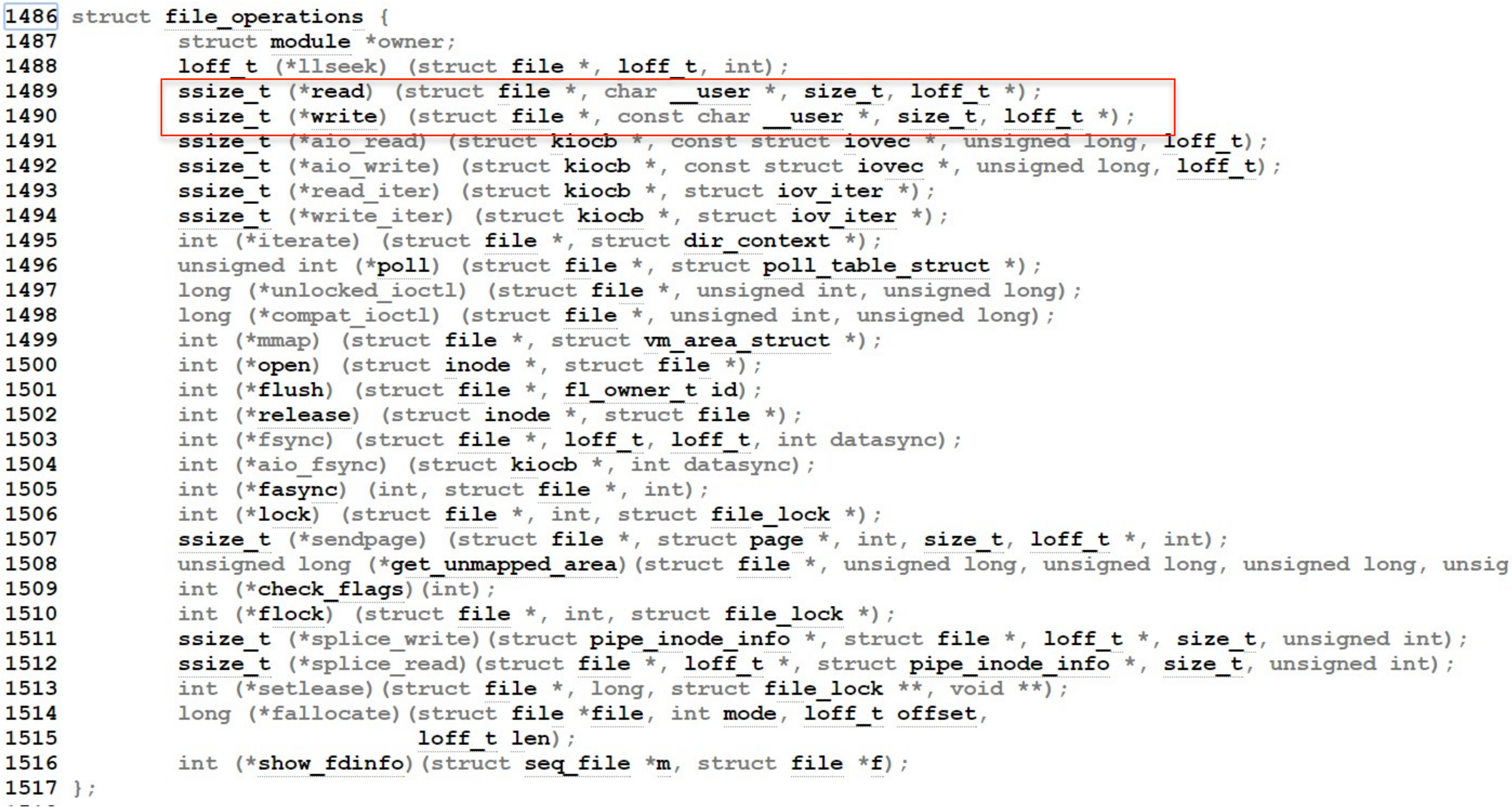


```
static inline struct proc_dir_entry *proc_create(
        const char *name, umode_t mode, struct proc_dir_entry *parent,
        const struct file_operations *proc_fops)
{
        return proc_create_data(name, mode, parent, proc_fops, NULL);
}

```

- Create a file under /proc
  - proc_create()

Using /proc in MP1



```
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, unsig
        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 *m, struct file *f);
};
```

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 ;
}
```

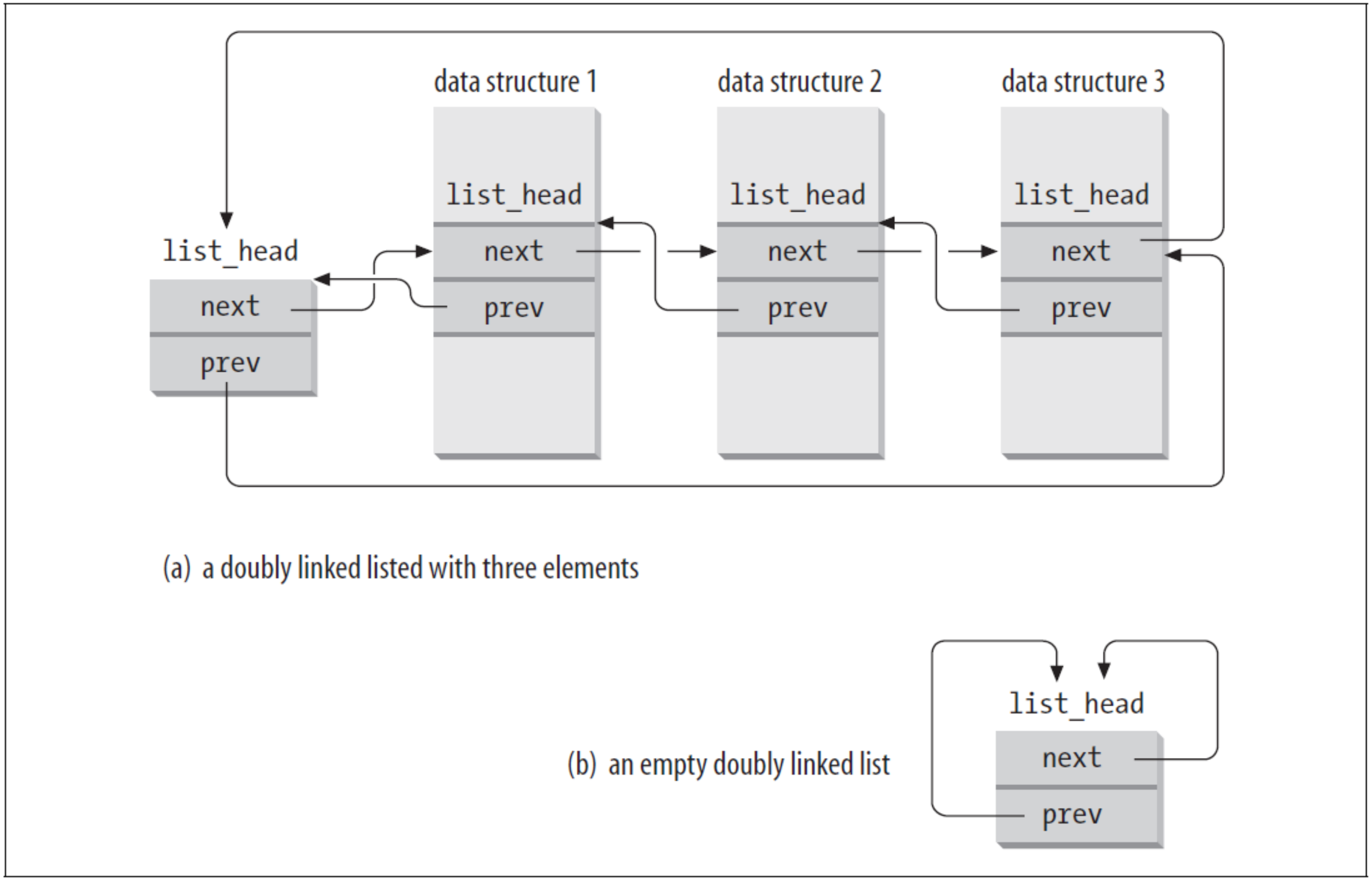
Linux Kernel List

- You will use Linux list to store all registered user processes
- Linux kernel list is a widely used data structure in Linux kernel
  - Define in <linux/linux.h>
  - You MUST get familiar of how to use it
  - Can be used as follows

```
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 List



(a) a doubly linked listed with three elements

(b) an empty doubly linked list

Figure 3-3. Doubly linked lists built with list_head data structures

Linux Kernel List

- Some useful APIs
  - 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 jiffies domain
  - msec_to_jiffies() to convert ms to jiffies
  - jiffies_to_msec() to convert jiffies to ms

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

- The expires field represents the jiffies value when the timer is expected to run

Kernel Timer

- Some useful API
  - 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);

Workqueue

- Allow kernel code to request that a function be called at some future time
  - Workqueue functions can sleep
  - Can be used to implement to bottom half of the interrupt handlers

- Some useful API
  - 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)

More questions?

- Office hours

- Piazza