CS 423
Operating System Design: Introduction to Linux Kernel Programming (MP1 Q&A)

Professor Adam Bates
Fall 2018

Goals for Today

- Learning Objectives:
  - Talk about the relevant skills required in MP1
- Announcements:
  - MP1 available on Compass2G. **Due February 19th!**
  - MP1 skeleton source now included on Compass (srry)
  - **Midterm Date/Time**: Wednesday, March 7th (in class)
  - **Final Date/Time**: Friday May 4th, 1:30pm - 4:30pm
  - Office Hours:

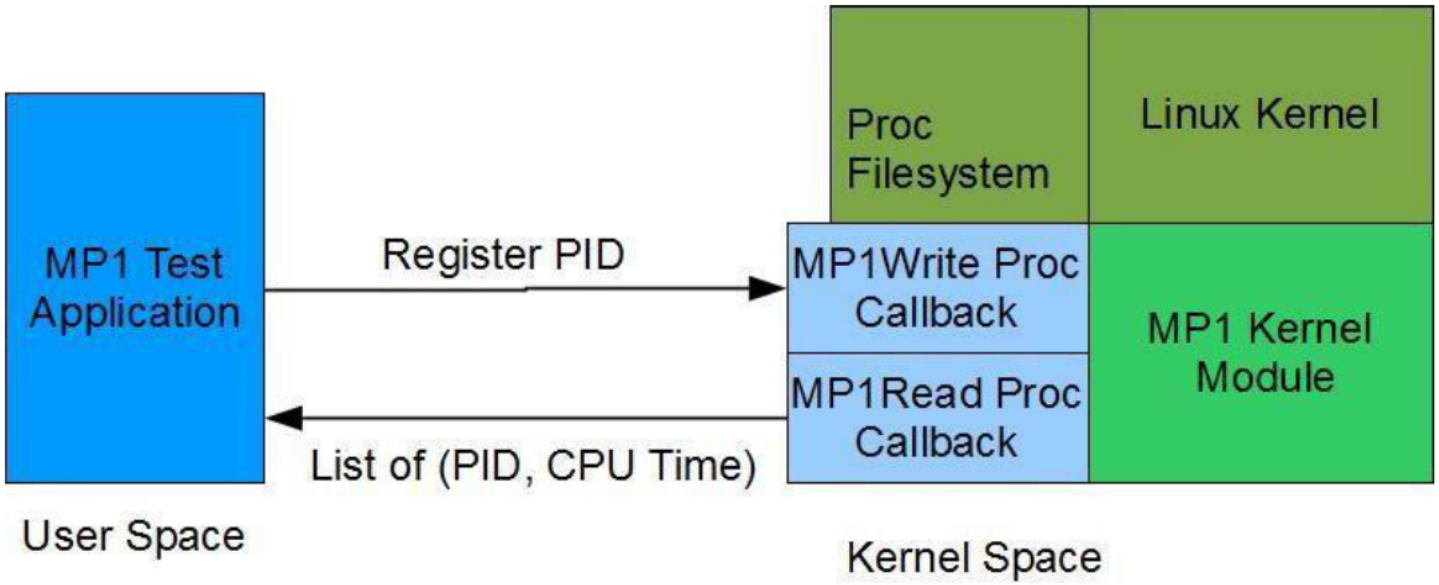
Adam: Tue 11am, Siebel 4306
Mohammad: Wed 5pm, Siebel 0207
Saad: Thur 3pm, Siebel 0207

Reminder: Please put away devices at the start of class

MP1 Goals

- Get yourself familiar with Linux kernel programming
- Learn to use the kernel's linked list data structure
- Learn to use proc FS to communicate between kernel and use space program
- Timers, workqueues, interrupts, 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 faults**
  - **Can conveniently Debug the program**
- Preemption
  - **Scheduling is not our responsibility**
- Signals (e.g., Ctrl+C)
- Libraries
- In Linux, everything is a file
- 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**
- Separate compilation
- Runtime linkage
- Entry and Exit functions

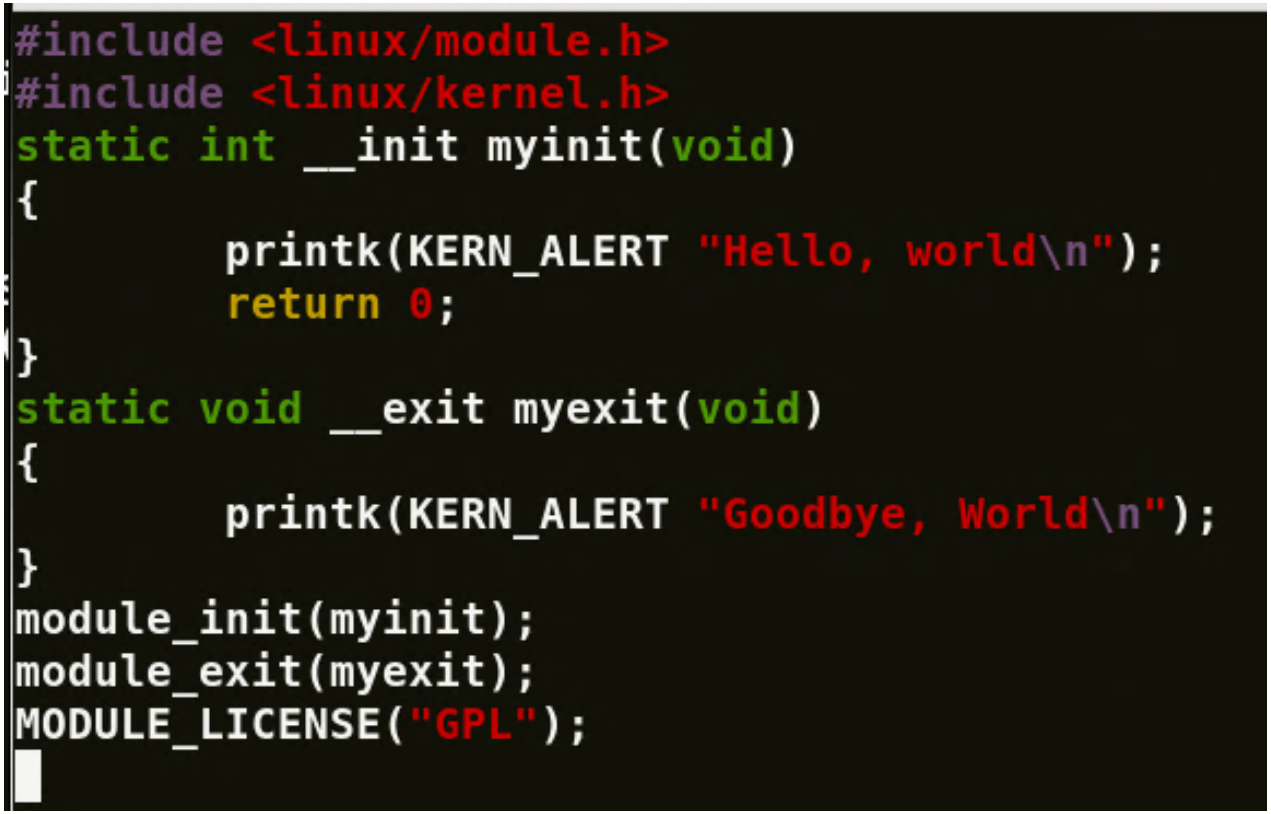
```
#include <linux/module.h>
#include <linux/kernel.h>

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");
```

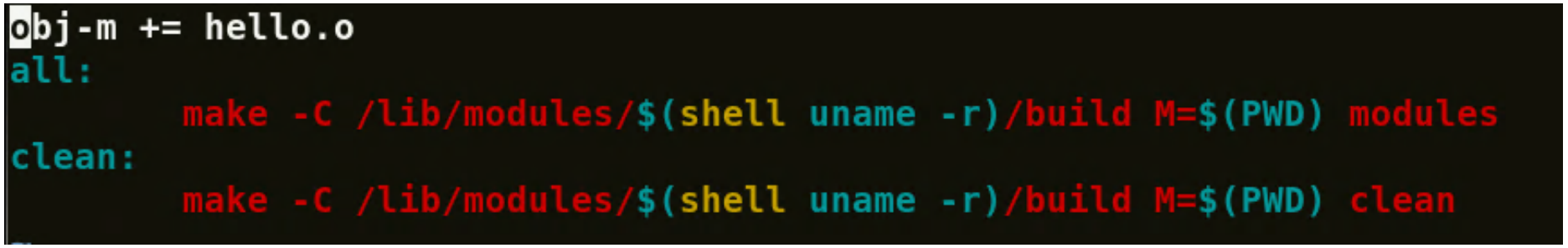
LKM "Hello World"



```c
#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

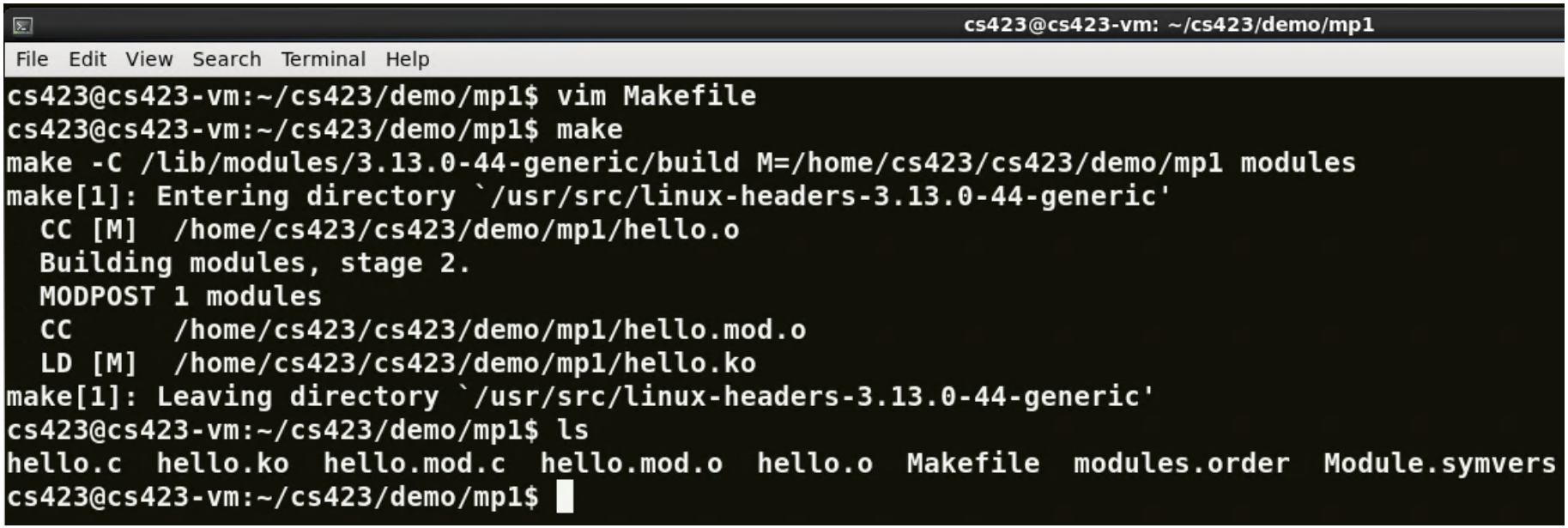
LKM "Hello World"



```
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
```

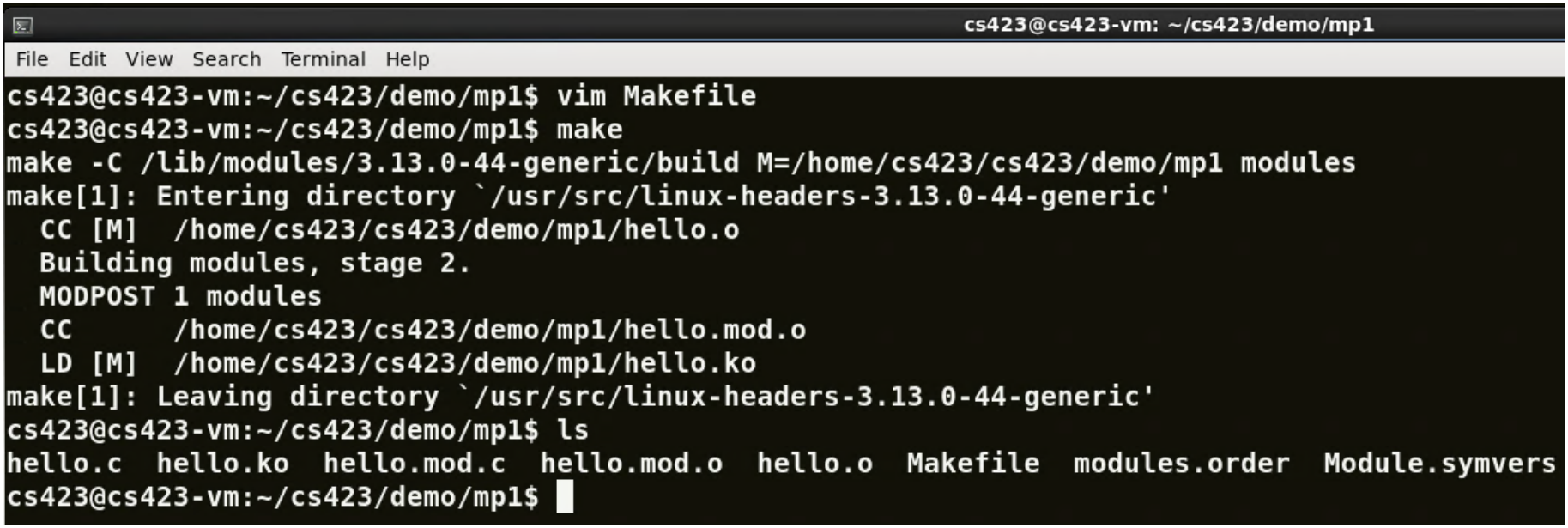
- Edit the Makefile
- For MP1, the Makefile is provided
  - It can be reused for MP2/MP3

LKM "Hello World"



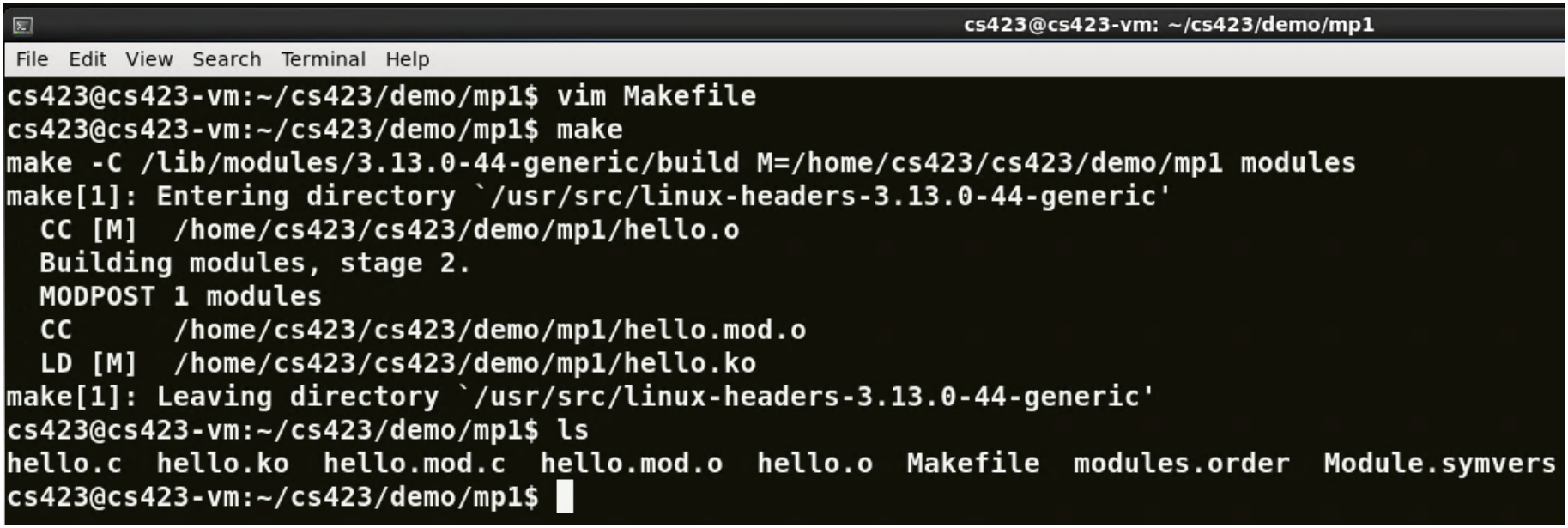
- Make
  - (Compiles the module)
- ls
  - Show module has been compiled to hello.ko

LKM "Hello World"



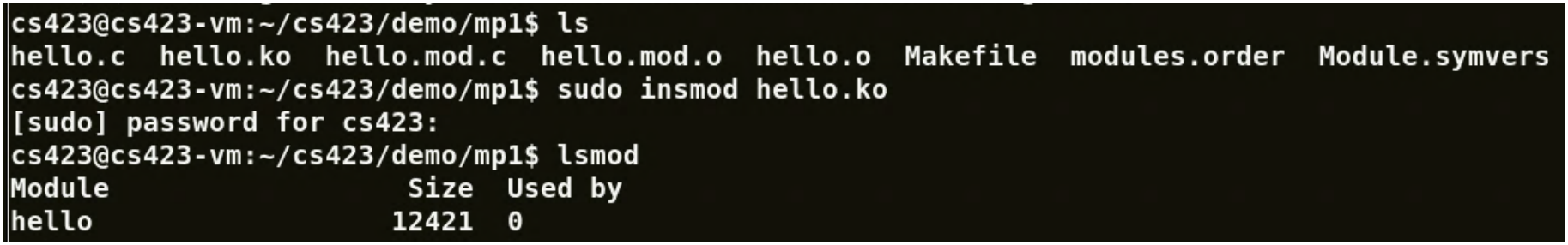
- Make
  - (Compiles the module)

LKM "Hello World"



- Make
  - (Compiles the module)
- ls
  - Show module has been compiled to 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
```

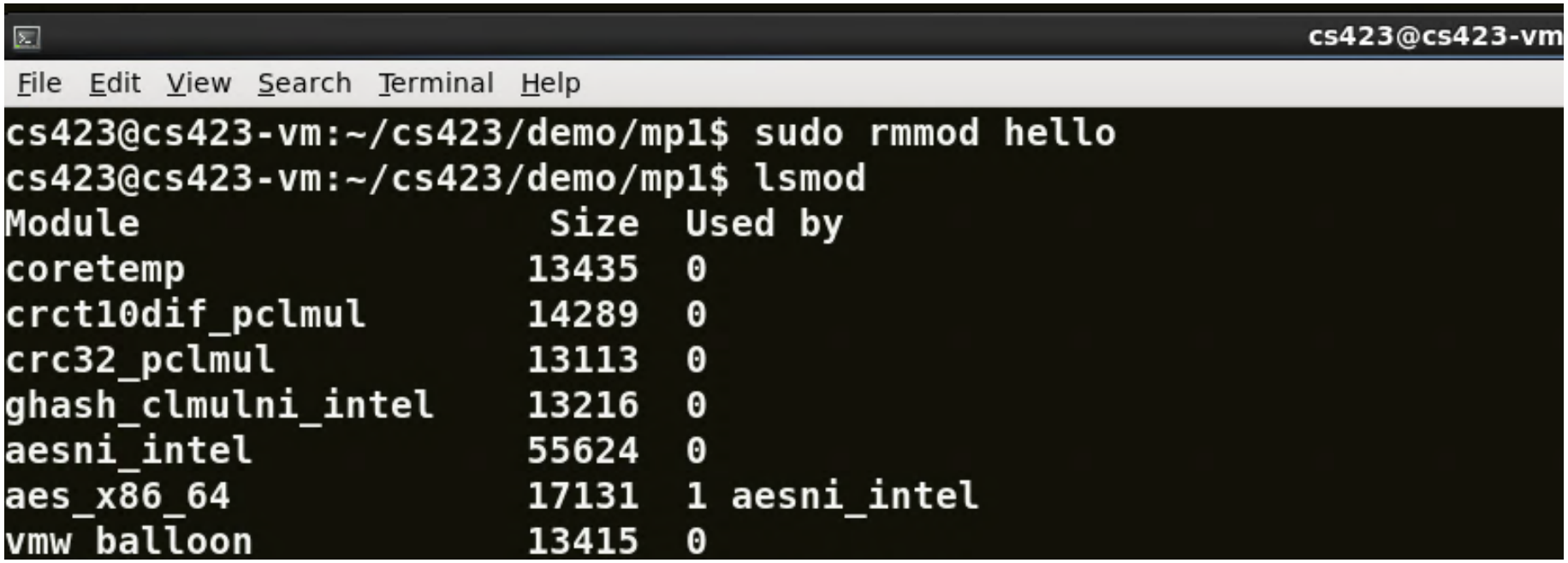
- sudo insmod hello.ko
  - (Installs the module)
- lsmod
  - Shows installed modules, including hello

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$
```

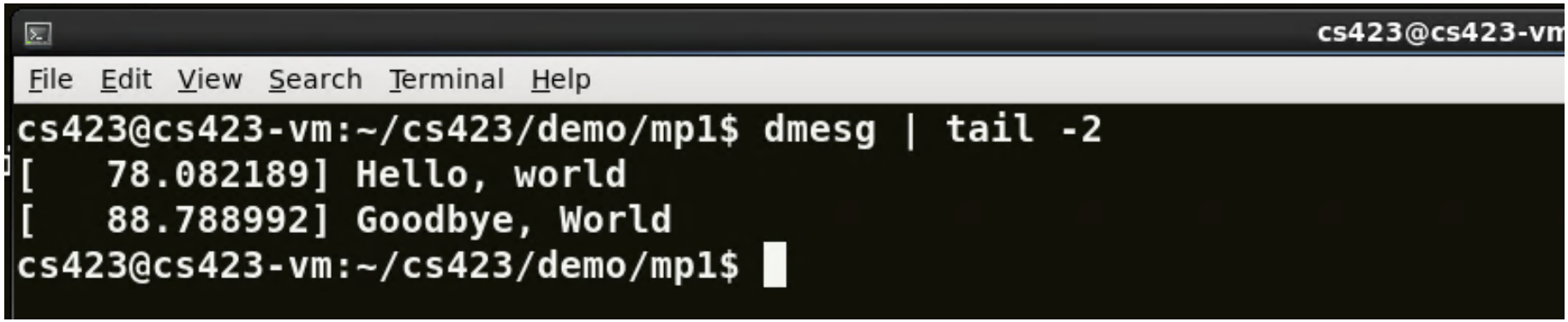
- modinfo
  - Lists the modules information

LKM "Hello World"



- sudo rmmod hello
  - Uninstalls the module

LKM "Hello World"



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

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 vs. Application Programming

Kernel Module (LKM)

- 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()

Application

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

Functions available to LKM

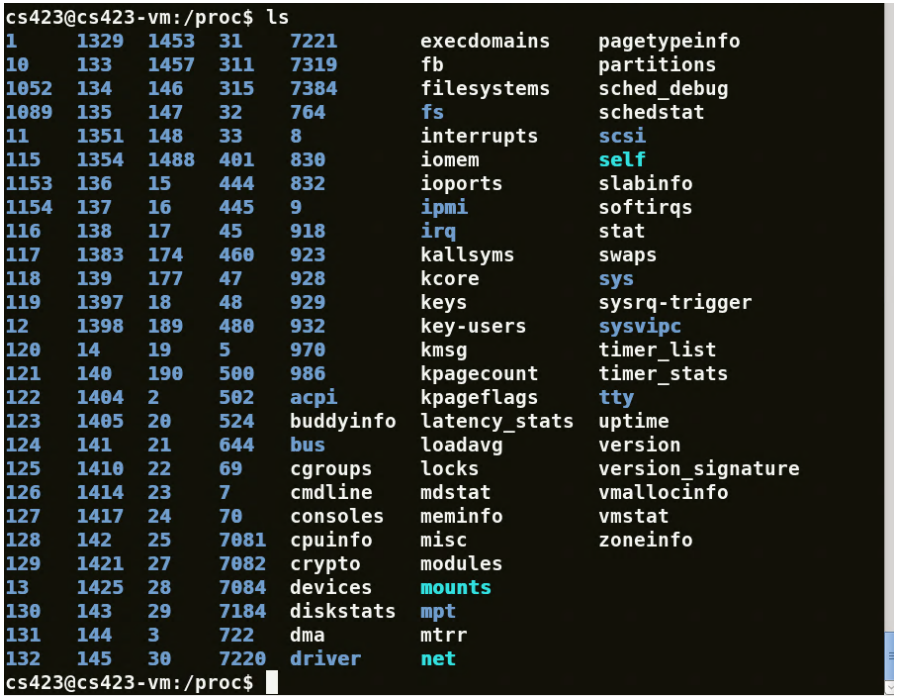
- Applications have access to library functions
  - printf(), malloc(), free()
- Kernel modules do not have access to library functions except those provided by kernel
  - printk(), kmalloc(), kfree(), vmalloc()
  - Check /proc/kallsyms to see a list of kernel provided functions
- Check Linux Kernel 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
  - e.g., `/proc/modules, /proc/meminfo, /proc/cpuinfo`

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

The /proc file system



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

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

Create a directory under /proc

proc_mkdir()

```
30 static inline struct proc_dir_entry *proc_create(
31         const char *name, umode_t mode, struct proc_dir_entry *parent,
32         const struct file_operations *proc_fops)
33 {
34         return proc_create_data(name, mode, parent, proc_fops, NULL);
35 }
36
```

Create a file under /proc

proc_create()

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, unsig
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
  - 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 Lists

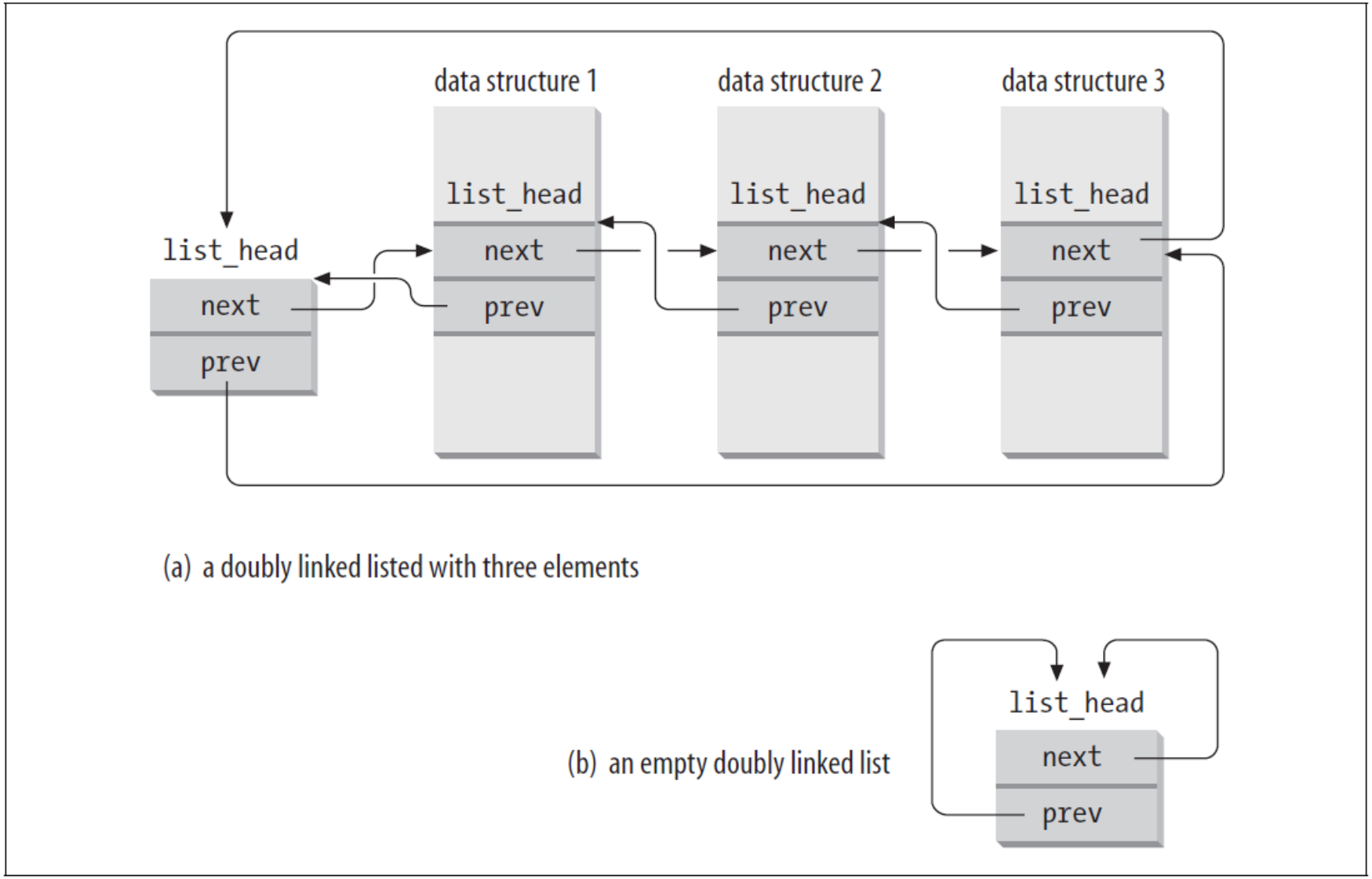


Figure 3-3. Doubly linked lists built with list_head data structures

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() 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 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

- 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 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!