CS 3502
Operating Systems

Project Lab

Kun Suo

Computer Science, Kennesaw State University

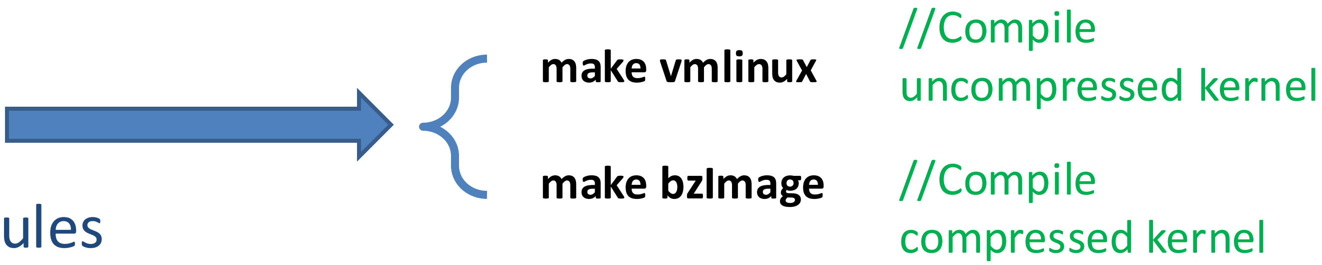
https://kevinsuo.github.io/

Outline

- Part 1: Create a helloworld kernel module (20')
- Part 2: Create an entry in the /proc file system for user-level read and write (30')
- Part 3: Exchange data between the user and kernel space via mmap (50')

Part 1: Kernel module

- Compile and install the Linux kernel
  - make
  - make modules
  - make modules_install
  - make install

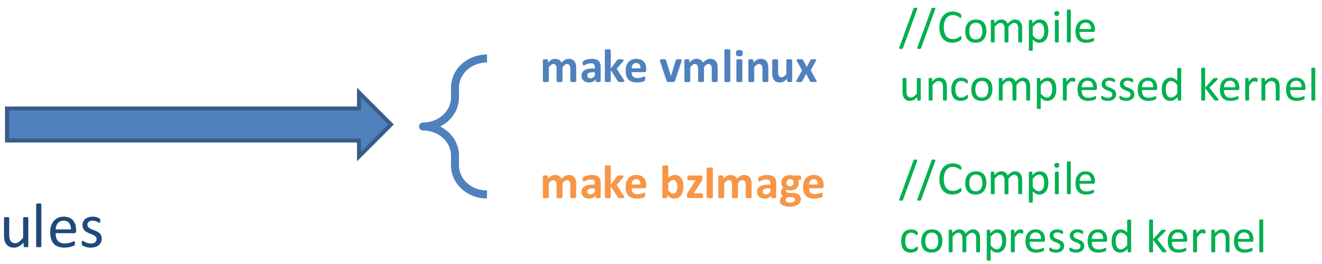


make vmlinux //Compile uncompressed kernel

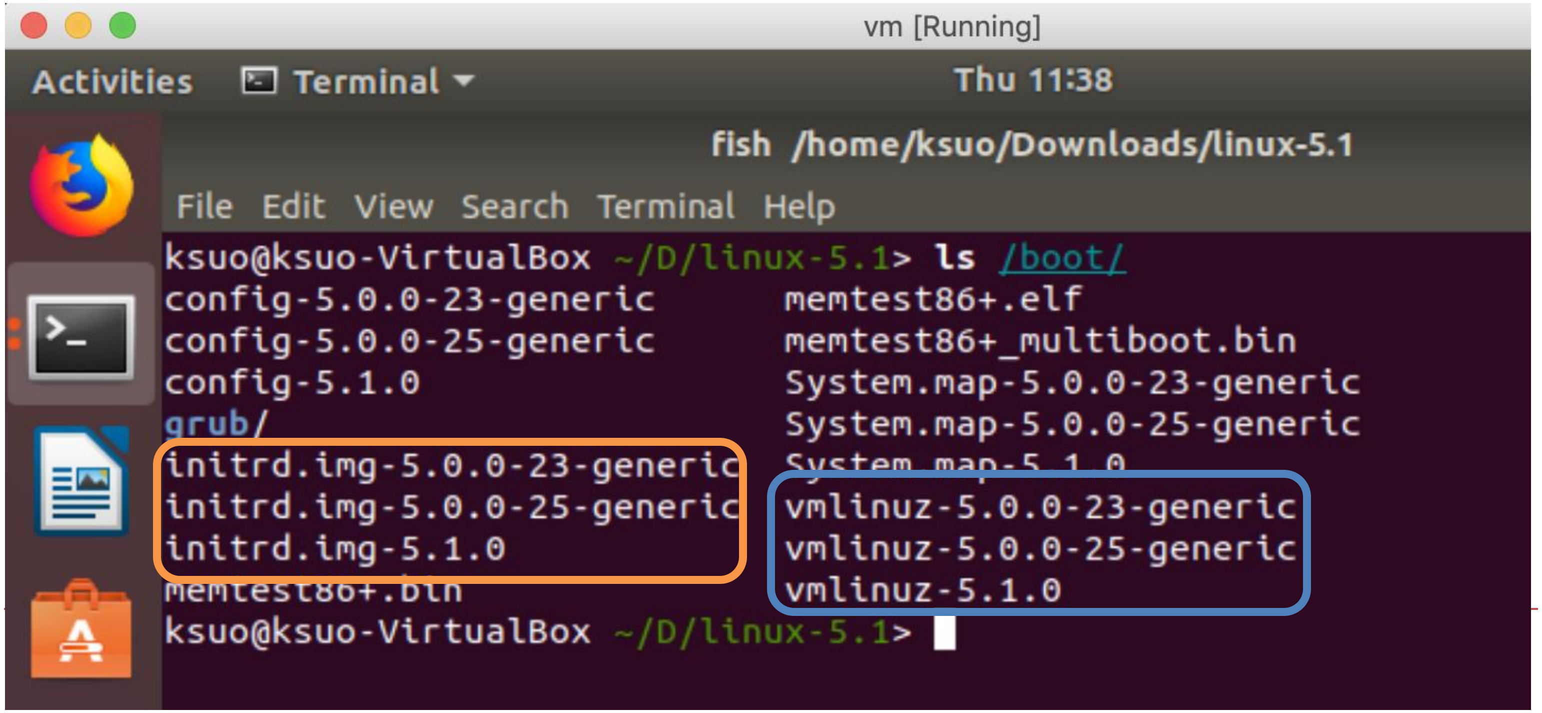
make bzImage //Compile compressed kernel

Part 1: Kernel module

- Compile and install the Linux kernel
  - make
  - make modules


make vmlinux //Compile uncompressed kernel

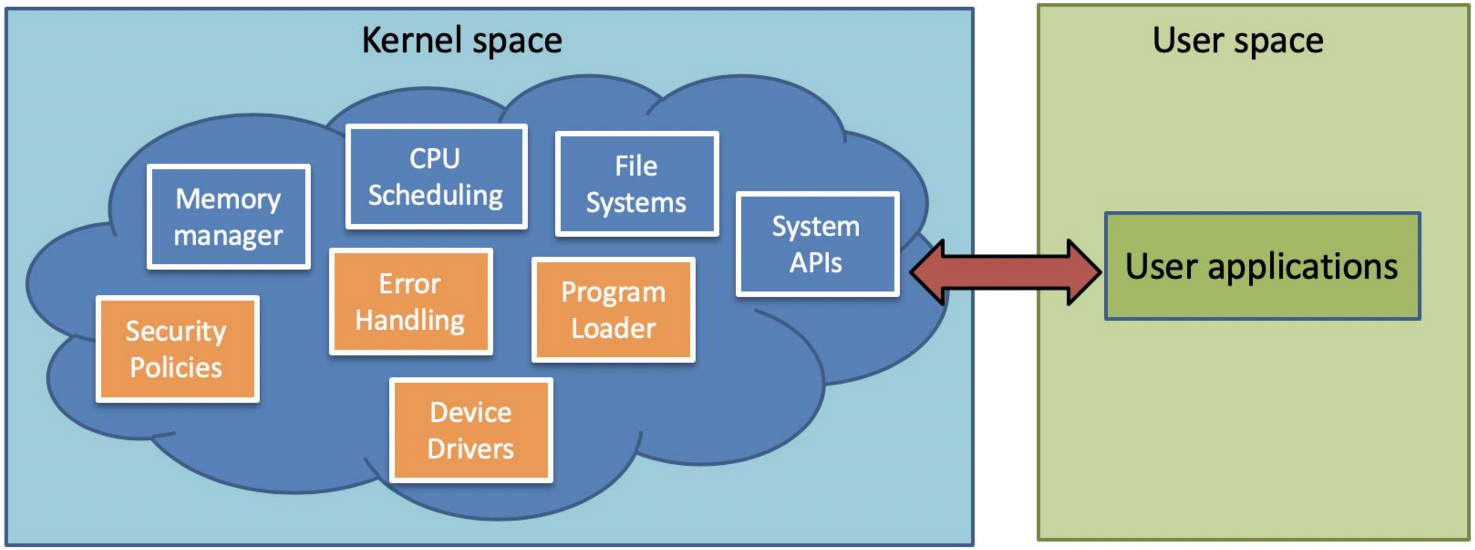
make bzImage //Compile compressed kernel


vm [Running]
Activities
Terminal
Thu 11:38
fish /home/ksuo/Downloads/linux-5.1
File Edit View Search Terminal Help

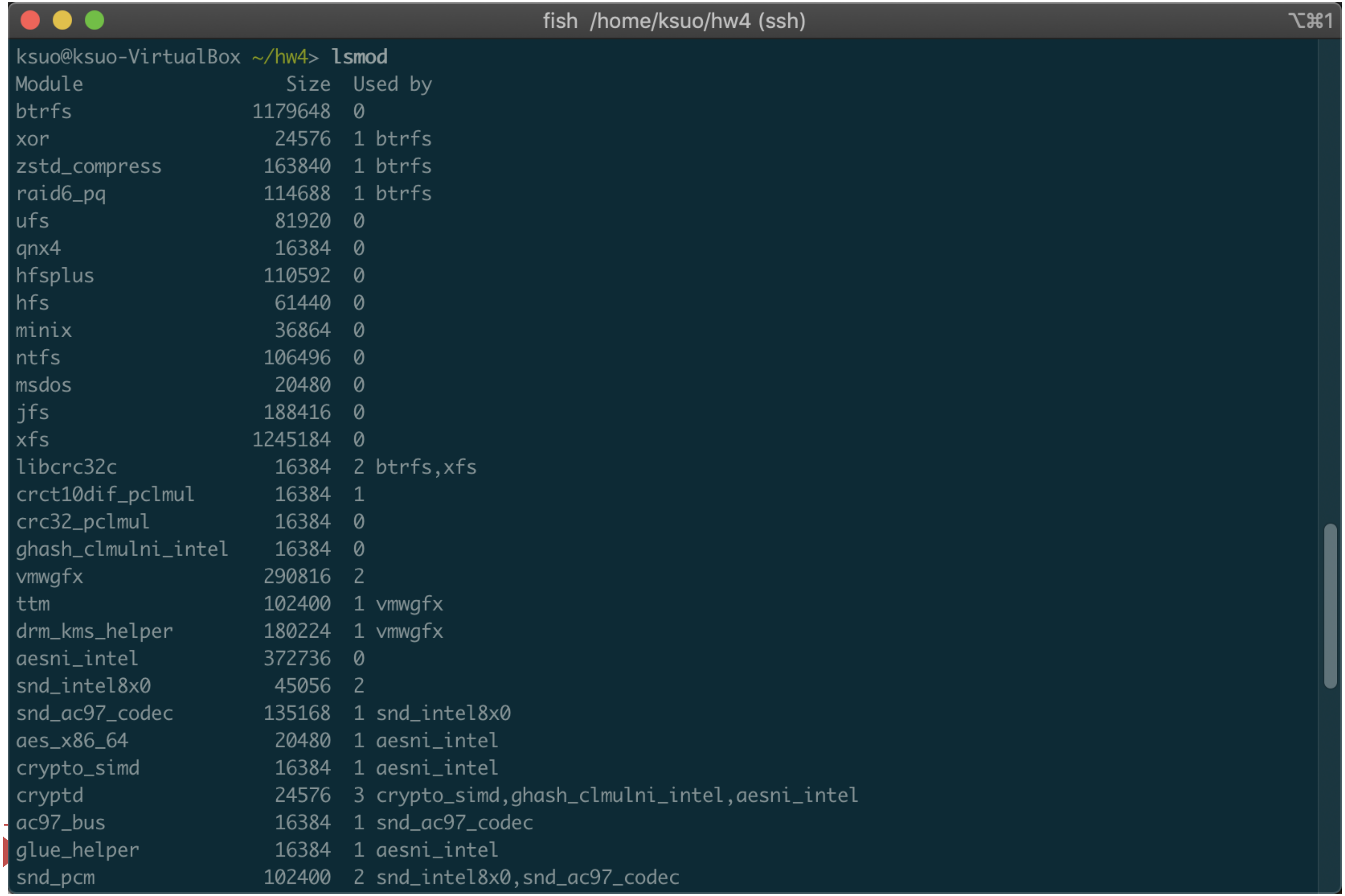
```
ksuo@ksuo-VirtualBox ~/D/linux-5.1> ls /boot/
config-5.0.0-23-generic     memtest86+.elf
config-5.0.0-25-generic     memtest86+_multiboot.bin
config-5.1.0                System.map-5.0.0-23-generic
grub/                       System.map-5.0.0-25-generic
initrd.img-5.0.0-23-generic System.map-5.1.0
initrd.img-5.0.0-25-generic vmlinuz-5.0.0-23-generic
initrd.img-5.1.0            vmlinuz-5.0.0-25-generic
memtest86+.bin              vmlinuz-5.1.0
ksuo@ksuo-VirtualBox ~/D/linux-5.1>
```

Part 1: Kernel module

- Compile and install the Linux kernel
  - make
  - make modules
  - make modules_install
  - make install



List all modules in the kernel

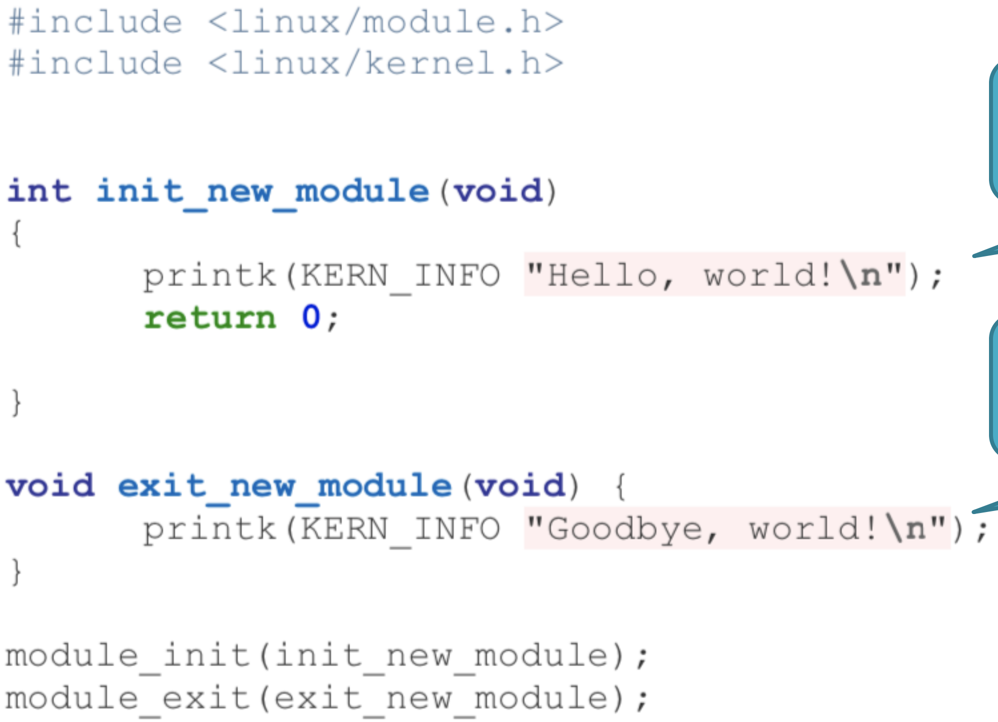


```
ksuo@ksuo-VirtualBox ~/hw4> lsmod
Module                  Size  Used by
btrfs                1179648  0
xor                    24576  1 btrfs
zstd_compress         163840  1 btrfs
raid6_pq              114688  1 btrfs
ufs                    81920  0
qnx4                   16384  0
hfsplus               110592  0
hfs                    61440  0
minix                  36864  0
ntfs                  106496  0
msdos                  20480  0
jfs                   188416  0
xfs                  1245184  0
libcrc32c              16384  2 btrfs,xfs
crct10dif_pclmul       16384  1
crc32_pclmul           16384  0
ghash_clmulni_intel    16384  0
vmwgfx                290816  2
ttm                   102400  1 vmwgfx
drm_kms_helper        180224  1 vmwgfx
aesni_intel           372736  0
snd_intel8x0           45056  2
snd_ac97_codec        135168  1 snd_intel8x0
aes_x86_64             20480  1 aesni_intel
crypto_simd            16384  1 aesni_intel
cryptd                 24576  3 crypto_simd,ghash_clmulni_intel,aesni_intel
ac97_bus               16384  1 snd_ac97_codec
glue_helper            16384  1 aesni_intel
snd_pcm               102400  2 snd_intel8x0,snd_ac97_codec
```

Build your module

https://github.com/kevinsuo/CS3502/blob/master/project-4-1.c

new_module.c



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


int init_new_module(void)
{
        printk(KERN_INFO "Hello, world!\n");
        return 0;

}

void exit_new_module(void) {
        printk(KERN_INFO "Goodbye, world!\n");
}

module_init(init_new_module);
module_exit(exit_new_module);
```

init_module is invoked when the module is loaded into the kernel

exit_module is called when the module is removed from the kernel

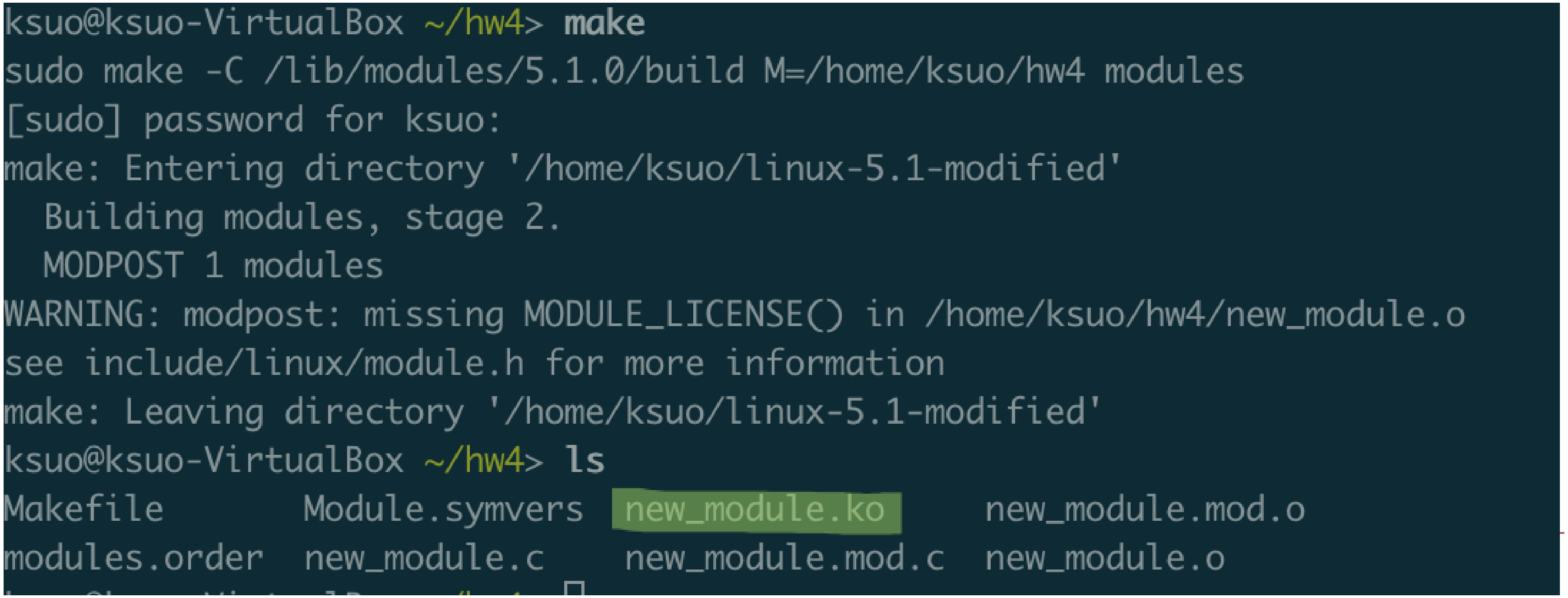
Compile your module

https://github.com/kevinsuo/CS3502/blob/master/project-4-1-Makefile

Makefile

new_module.o is the output file

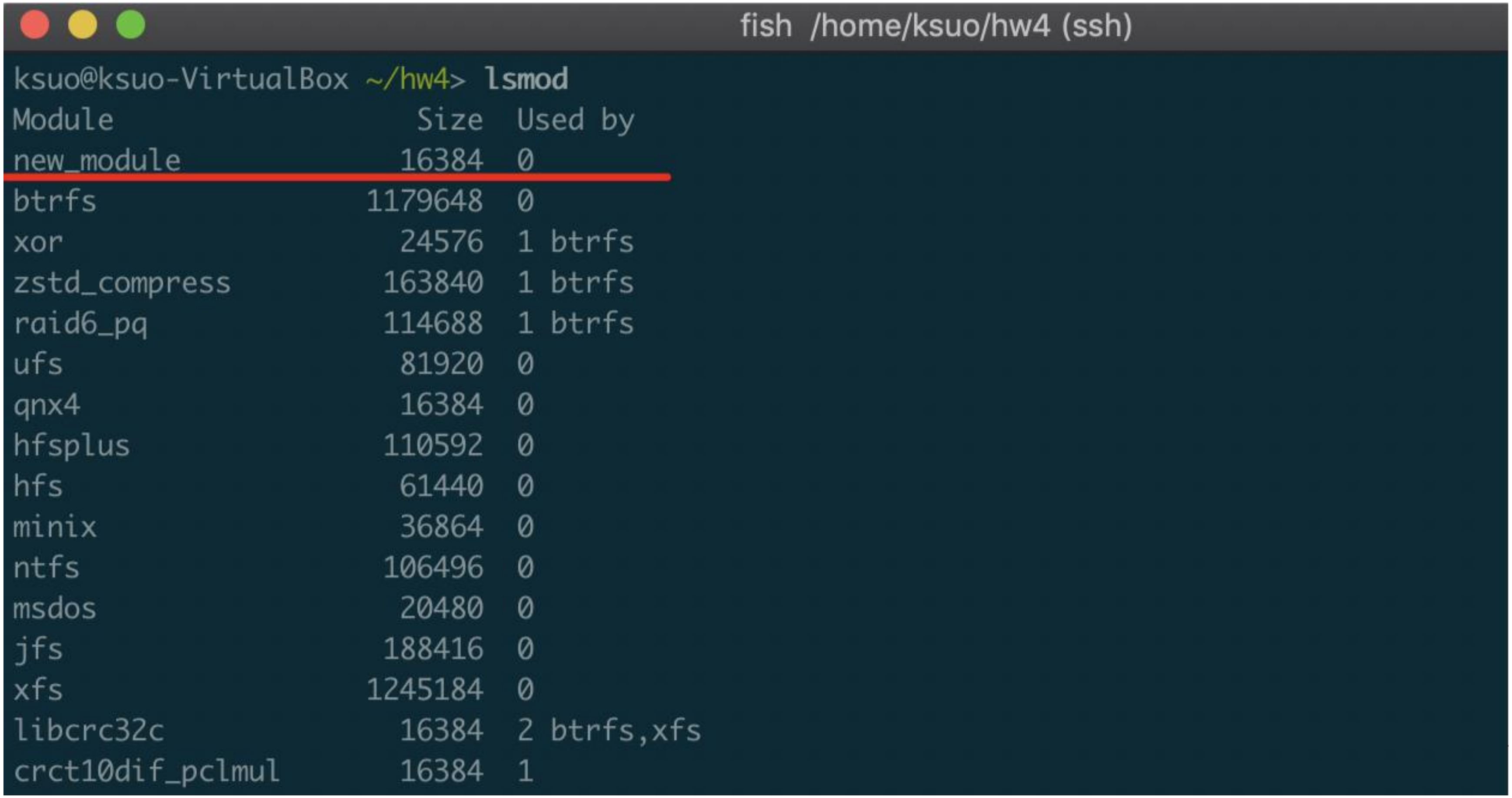
```
obj-m += new_module.o
all:
	sudo make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
	sudo make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```



```
ksuo@ksuo-VirtualBox ~/hw4> make
sudo make -C /lib/modules/5.1.0/build M=/home/ksuo/hw4 modules
[sudo] password for ksuo:
make: Entering directory '/home/ksuo/linux-5.1-modified'
  Building modules, stage 2.
  MODPOST 1 modules
WARNING: modpost: missing MODULE_LICENSE() in /home/ksuo/hw4/new_module.o
see include/linux/module.h for more information
make: Leaving directory '/home/ksuo/linux-5.1-modified'
ksuo@ksuo-VirtualBox ~/hw4> ls
Makefile       Module.symvers  new_module.ko      new_module.mod.o
modules.order  new_module.c    new_module.mod.c   new_module.o
```

Insert the module into the Linux kernel

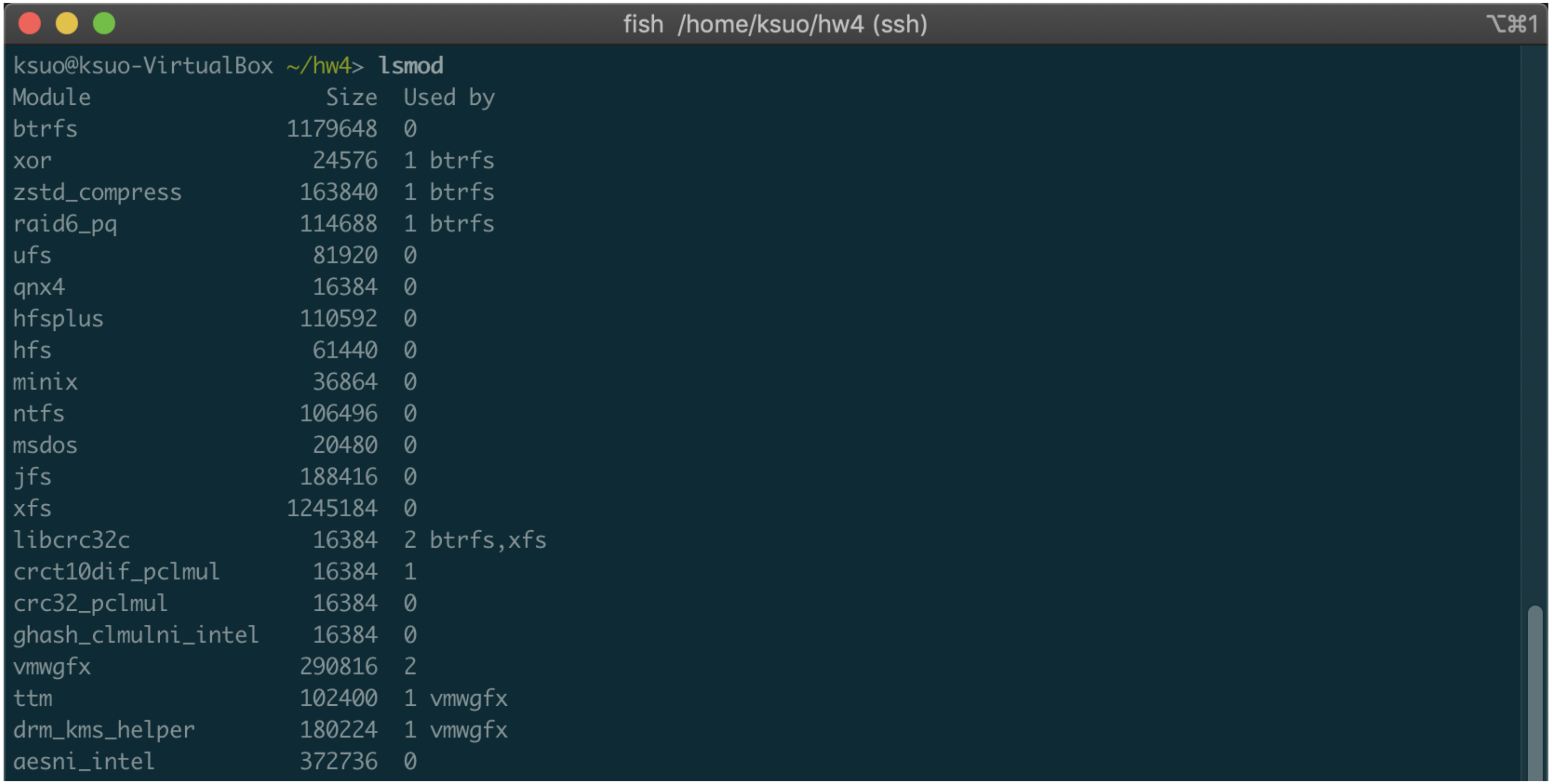
- sudo insmod new_module.ko



```
ksuo@ksuo-VirtualBox ~/hw4> lsmod
Module                  Size  Used by
new_module             16384  0
btrfs                1179648  0
xor                    24576  1 btrfs
zstd_compress         163840  1 btrfs
raid6_pq              114688  1 btrfs
ufs                    81920  0
qnx4                   16384  0
hfsplus               110592  0
hfs                    61440  0
minix                  36864  0
ntfs                  106496  0
msdos                  20480  0
jfs                   188416  0
xfs                  1245184  0
libcrc32c              16384  2 btrfs,xfs
crct10dif_pclmul       16384  1
```

Remove the module from the kernel

- sudo rmmod new_module



```
ksuo@ksuo-VirtualBox ~/hw4> lsmod
Module                  Size  Used by
btrfs                1179648  0
xor                    24576  1 btrfs
zstd_compress         163840  1 btrfs
raid6_pq              114688  1 btrfs
ufs                    81920  0
qnx4                   16384  0
hfsplus               110592  0
hfs                    61440  0
minix                  36864  0
ntfs                  106496  0
msdos                  20480  0
jfs                   188416  0
xfs                  1245184  0
libcrc32c              16384  2 btrfs,xfs
crct10dif_pclmul       16384  1
crc32_pclmul           16384  0
ghash_clmulni_intel    16384  0
vmwgfx                290816  2
ttm                   102400  1 vmwgfx
drm_kms_helper        180224  1 vmwgfx
aesni_intel           372736  0
```

Related codes

- ***new_module.c:*** https://github.com/kevinsuo/CS3502/blob/master/project-4-1.c

- ***Makefile:*** https://github.com/kevinsuo/CS3502/blob/master/project-4-1-Makefile

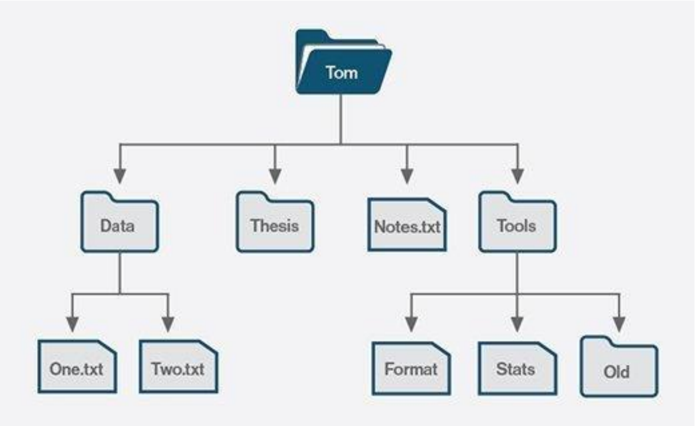
Outline

- Part 1: Create a helloworld kernel module (20')
- Part 2: Create an entry in the /proc file system for user level read and write (30')
- Part 3: Exchange data between the user and kernel space via mmap (50')

File System

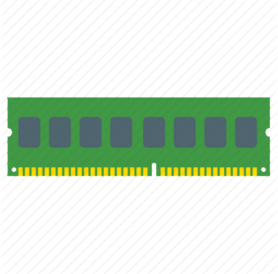
OS

File system



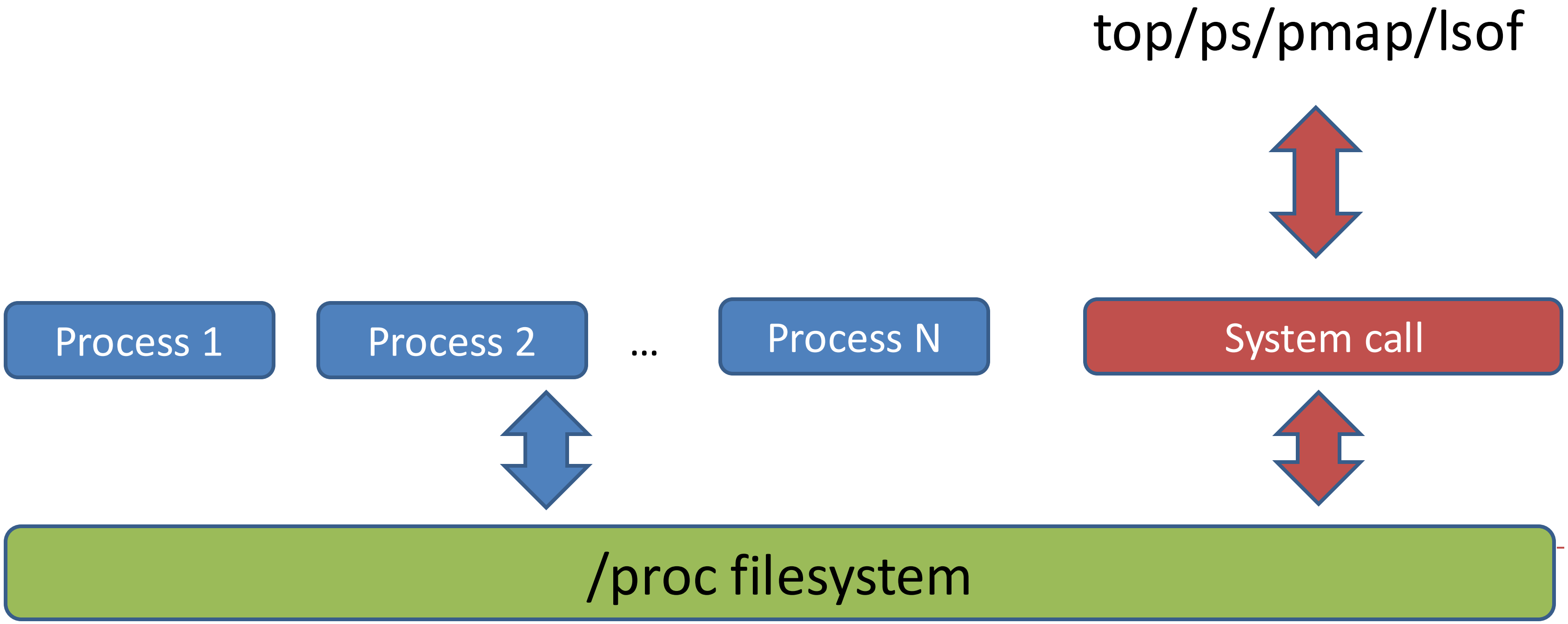
Proc File system

Virtual
FS

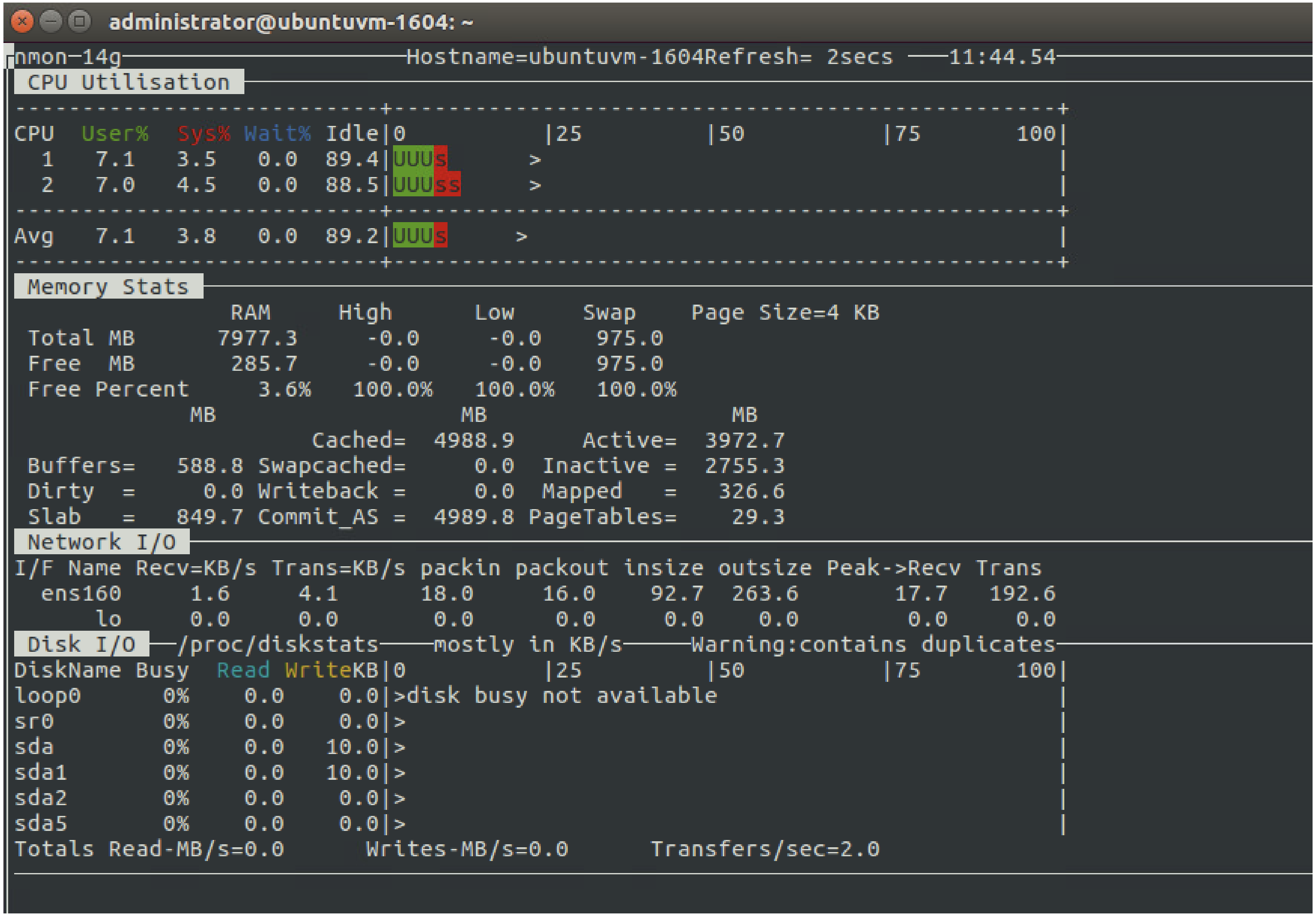


Proc file system

- The proc filesystem (procfs) is a special filesystem in Unix-like OS that presents information about processes and other system information in a hierarchical file-like structure, providing a convenient and standardized method for dynamically accessing process data held in the kernel



Proc file system

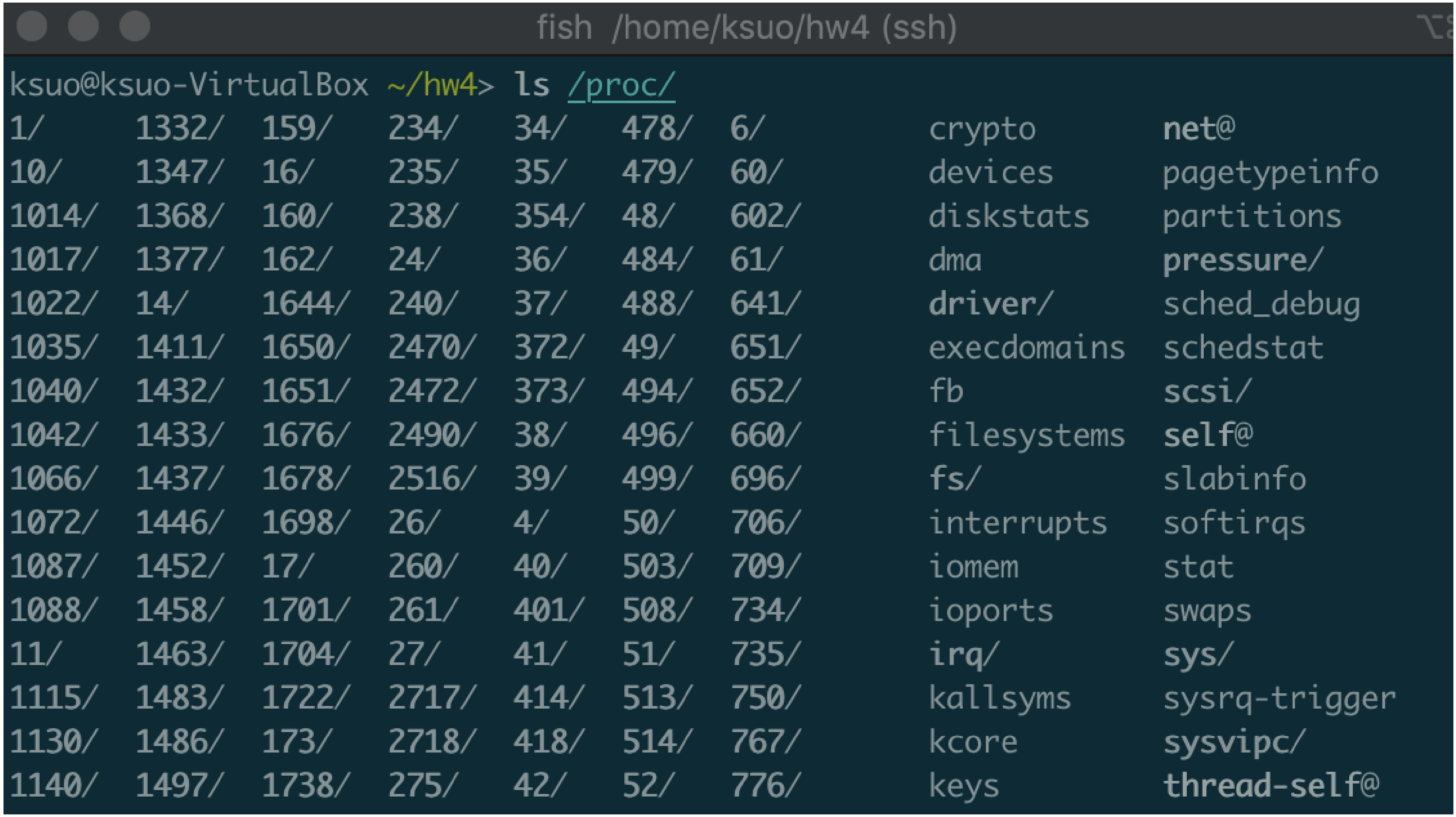


administrator@ubuntuvm-1604: ~

```
nmon─14g─────────────────────Hostname=ubuntuvm-1604Refresh= 2secs ───11:44.54─────
 CPU Utilisation ─────────────────────────────────────────────────────────────────
---------------------------+-------------------------------------------------+
CPU  User%  Sys% Wait% Idle|0          |25         |50          |75       100|
  1   7.1   3.5   0.0  89.4|UUUs       >                                     |
  2   7.0   4.5   0.0  88.5|UUUss      >                                     |
---------------------------+-------------------------------------------------+
Avg   7.1   3.8   0.0  89.2|UUUs      >                                      |
---------------------------+-------------------------------------------------+
 Memory Stats ────────────────────────────────────────────────────────────────────
                RAM     High      Low      Swap    Page Size=4 KB
 Total MB      7977.3     -0.0     -0.0     975.0
 Free  MB       285.7     -0.0     -0.0     975.0
 Free Percent     3.6%  100.0%   100.0%   100.0%
             MB                  MB                  MB
                  Cached=  4988.9     Active=  3972.7
 Buffers=   588.8 Swapcached=   0.0  Inactive =  2755.3
 Dirty  =     0.0 Writeback =   0.0  Mapped   =   326.6
 Slab   =   849.7 Commit_AS =  4989.8 PageTables=    29.3
 Network I/O ─────────────────────────────────────────────────────────────────────
I/F Name Recv=KB/s Trans=KB/s packin packout insize outsize Peak->Recv Trans
  ens160     1.6      4.1      18.0     16.0    92.7  263.6       17.7   192.6
      lo     0.0      0.0       0.0      0.0     0.0    0.0        0.0     0.0
 Disk I/O ──/proc/diskstats────mostly in KB/s─────Warning:contains duplicates─────
DiskName Busy  Read WriteKB|0          |25         |50          |75       100|
loop0       0%    0.0    0.0|>disk busy not available                        |
sr0         0%    0.0    0.0|>                                               |
sda         0%    0.0   10.0|>                                               |
sda1        0%    0.0   10.0|>                                               |
sda2        0%    0.0    0.0|>                                               |
sda5        0%    0.0    0.0|>                                               |
Totals Read-MB/s=0.0      Writes-MB/s=0.0      Transfers/sec=2.0
──────────────────────────────────────────────────────────────────────────────────
```

Proc file system

- ls /proc/



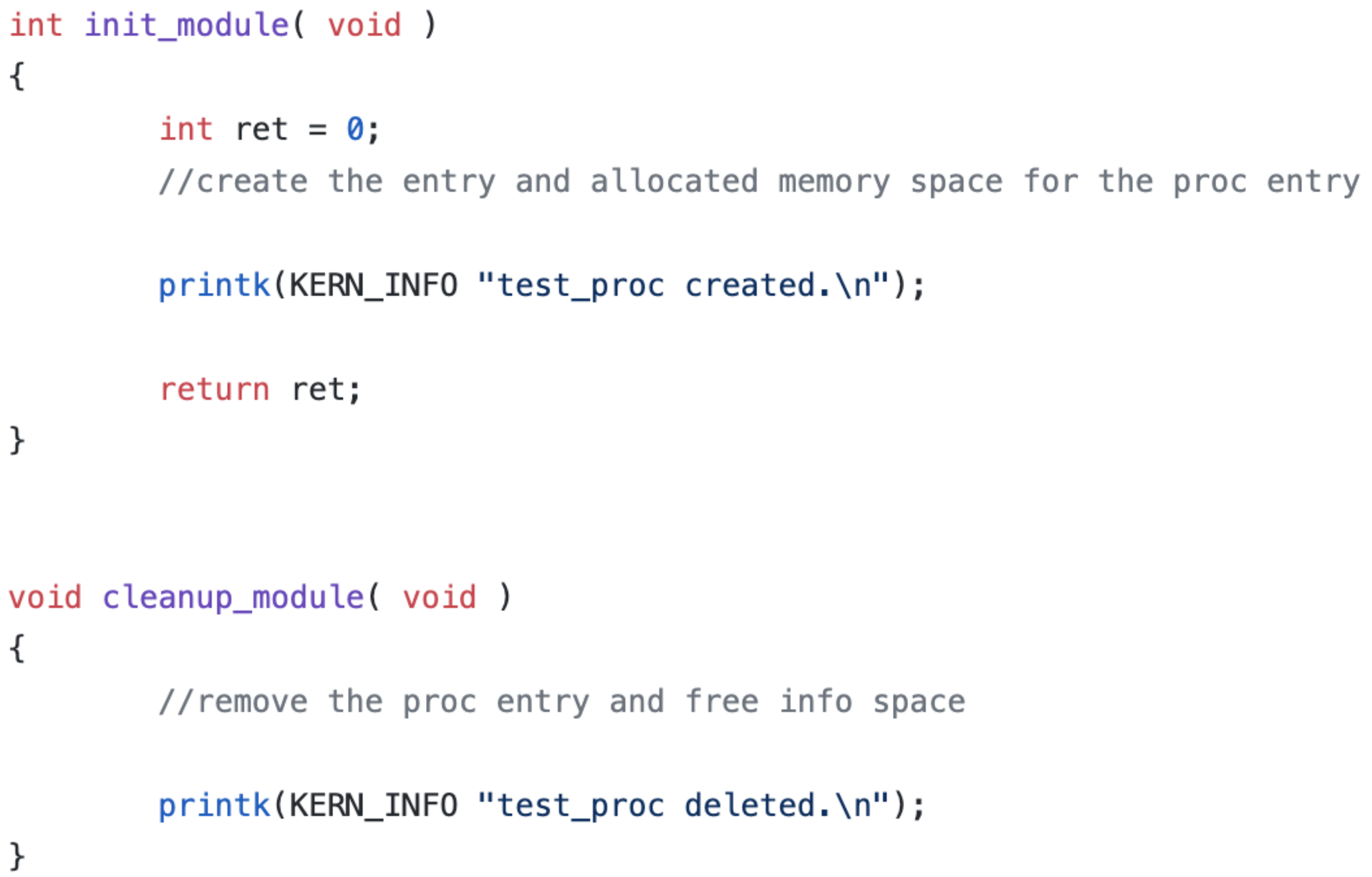
```
ksuo@ksuo-VirtualBox ~/hw4> ls /proc/
1/      1332/  159/   234/   34/   478/  6/          crypto        net@
10/     1347/  16/    235/   35/   479/  60/         devices       pagetypeinfo
1014/   1368/  160/   238/   354/  48/   602/        diskstats     partitions
1017/   1377/  162/   24/    36/   484/  61/         dma           pressure/
1022/   14/    1644/  240/   37/   488/  641/        driver/       sched_debug
1035/   1411/  1650/  2470/  372/  49/   651/        execdomains   schedstat
1040/   1432/  1651/  2472/  373/  494/  652/        fb            scsi/
1042/   1433/  1676/  2490/  38/   496/  660/        filesystems   self@
1066/   1437/  1678/  2516/  39/   499/  696/        fs/           slabinfo
1072/   1446/  1698/  26/    4/    50/   706/        interrupts    softirqs
1087/   1452/  17/    260/   40/   503/  709/        iomem         stat
1088/   1458/  1701/  261/   401/  508/  734/        ioports       swaps
11/     1463/  1704/  27/    41/   51/   735/        irq/          sys/
1115/   1483/  1722/  2717/  414/  513/  750/        kallsyms      sysrq-trigger
1130/   1486/  173/   2718/  418/  514/  767/        kcore         sysvipc/
1140/   1497/  1738/  275/   42/   52/   776/        keys          thread-self@
```

Proc file system

- /proc/cmdline – Kernel command line information.
- /proc/console – Information about current consoles including tty.
- /proc/devices – Device drivers currently configured for the running kernel.
- /proc/dma – Info about current DMA channels.
- /proc/fb – Framebuffer devices.
- /proc/filesystems – Current filesystems supported by the kernel.
- /proc/iomem – Current system memory map for devices.
- /proc/ioports – Registered port regions for input output communication with device.
- /proc/loadavg – System load average.
- /proc/locks – Files currently locked by kernel.
- /proc/meminfo – Info about system memory (see above example).
- /proc/misc – Miscellaneous drivers registered for miscellaneous major device.
- /proc/modules – Currently loaded kernel modules.
- /proc/mounts – List of all mounts in use by system.
- /proc/partitions – Detailed info about partitions available to the system.
- /proc/pci – Information about every PCI device.
- /proc/stat – Record or various statistics kept from last reboot.
- /proc/swap – Information about swap space.
- /proc/uptime – Uptime information (in seconds).
- /proc/version – Kernel version, gcc version, and Linux distribution installed.

Part 2: Create an entry in the /proc file system

- https://github.com/kevinsuo/CS3502/blob/master/project-4-2.c

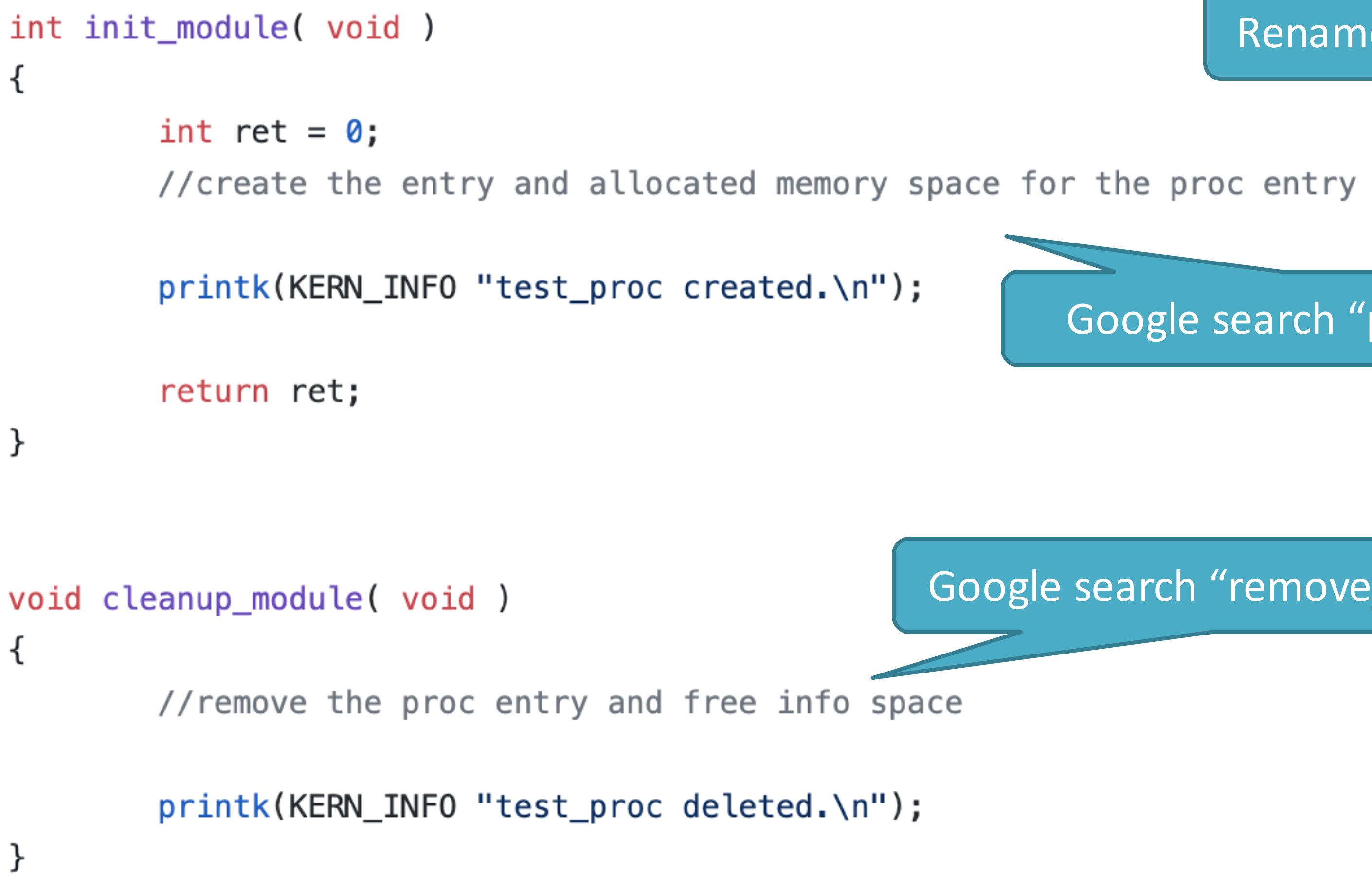


```c
int init_module( void )
{
        int ret = 0;
        //create the entry and allocated memory space for the proc entry

        printk(KERN_INFO "test_proc created.\n");

        return ret;
}


void cleanup_module( void )
{
        //remove the proc entry and free info space

        printk(KERN_INFO "test_proc deleted.\n");
}
```

Part 2: Create an entry in the /proc file system

- https://github.com/kevinsuo/CS3502/blob/master/project-4-2.c

Renamed as “myproc.c”

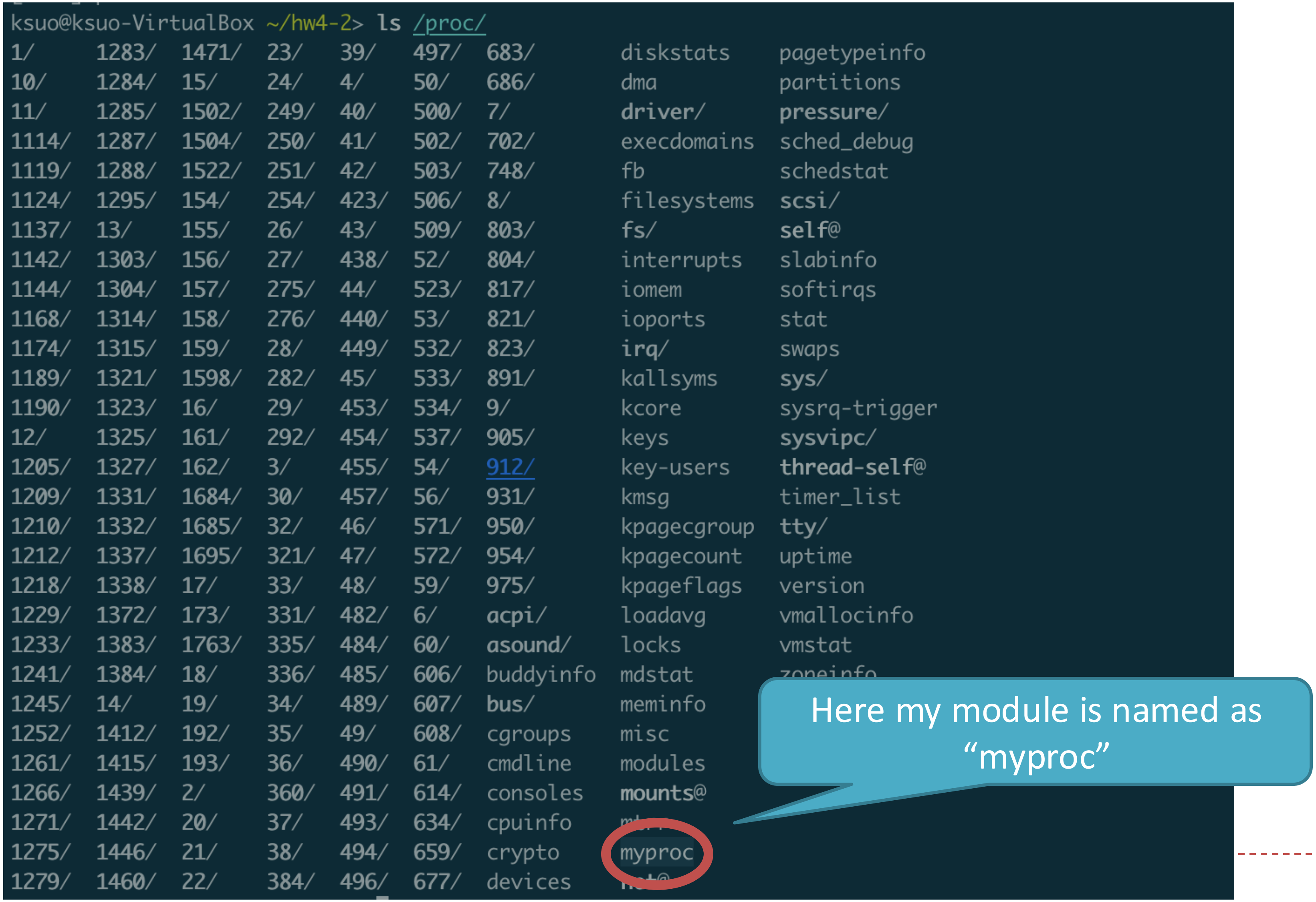


```
int init_module( void )
{
        int ret = 0;
        //create the entry and allocated memory space for the proc entry

        printk(KERN_INFO "test_proc created.\n");

        return ret;
}


void cleanup_module( void )
{
        //remove the proc entry and free info space

        printk(KERN_INFO "test_proc deleted.\n");
}
```

Google search “proc_create”

Google search “remove_proc_entry”

Part 2: after you insert your module, check whether it exists under /proc

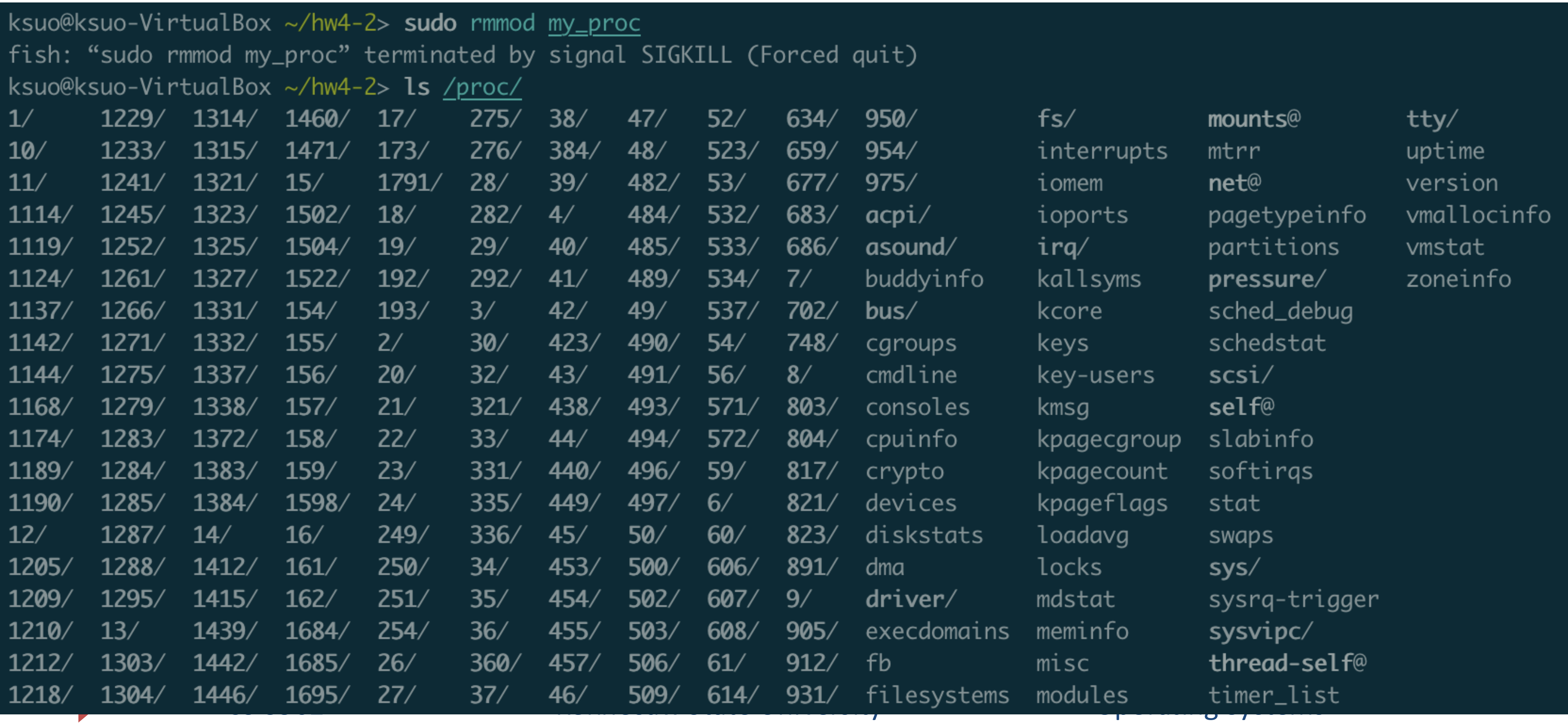


```
ksuo@ksuo-VirtualBox ~/hw4-2> ls /proc/
1/     1283/  1471/  23/    39/   497/  683/        diskstats    pagetypeinfo
10/    1284/  15/    24/    4/    50/   686/        dma          partitions
11/    1285/  1502/  249/   40/   500/  7/          driver/      pressure/
1114/  1287/  1504/  250/   41/   502/  702/        execdomains  sched_debug
1119/  1288/  1522/  251/   42/   503/  748/        fb           schedstat
1124/  1295/  154/   254/   423/  506/  8/          filesystems  scsi/
1137/  13/    155/   26/    43/   509/  803/        fs/          self@
1142/  1303/  156/   27/    438/  52/   804/        interrupts   slabinfo
1144/  1304/  157/   275/   44/   523/  817/        iomem        softirqs
1168/  1314/  158/   276/   440/  53/   821/        ioports      stat
1174/  1315/  159/   28/    449/  532/  823/        irq/         swaps
1189/  1321/  1598/  282/   45/   533/  891/        kallsyms     sys/
1190/  1323/  16/    29/    453/  534/  9/          kcore        sysrq-trigger
12/    1325/  161/   292/   454/  537/  905/        keys         sysvipc/
1205/  1327/  162/   3/     455/  54/   912/        key-users    thread-self@
1209/  1331/  1684/  30/    457/  56/   931/        kmsg         timer_list
1210/  1332/  1685/  32/    46/   571/  950/        kpagecgroup  tty/
1212/  1337/  1695/  321/   47/   572/  954/        kpagecount   uptime
1218/  1338/  17/    33/    48/   59/   975/        kpageflags   version
1229/  1372/  173/   331/   482/  6/    acpi/       loadavg      vmallocinfo
1233/  1383/  1763/  335/   484/  60/   asound/     locks        vmstat
1241/  1384/  18/    336/   485/  606/  buddyinfo   mdstat       zoneinfo
1245/  14/    19/    34/    489/  607/  bus/        meminfo
1252/  1412/  192/   35/    49/   608/  cgroups     misc
1261/  1415/  193/   36/    490/  61/   cmdline     modules
1266/  1439/  2/     360/   491/  614/  consoles    mounts@
1271/  1442/  20/    37/    493/  634/  cpuinfo     mtrr
1275/  1446/  21/    38/    494/  659/  crypto      myproc
1279/  1460/  22/    384/   496/  677/  devices     net@
```

Here my module is named as “myproc”

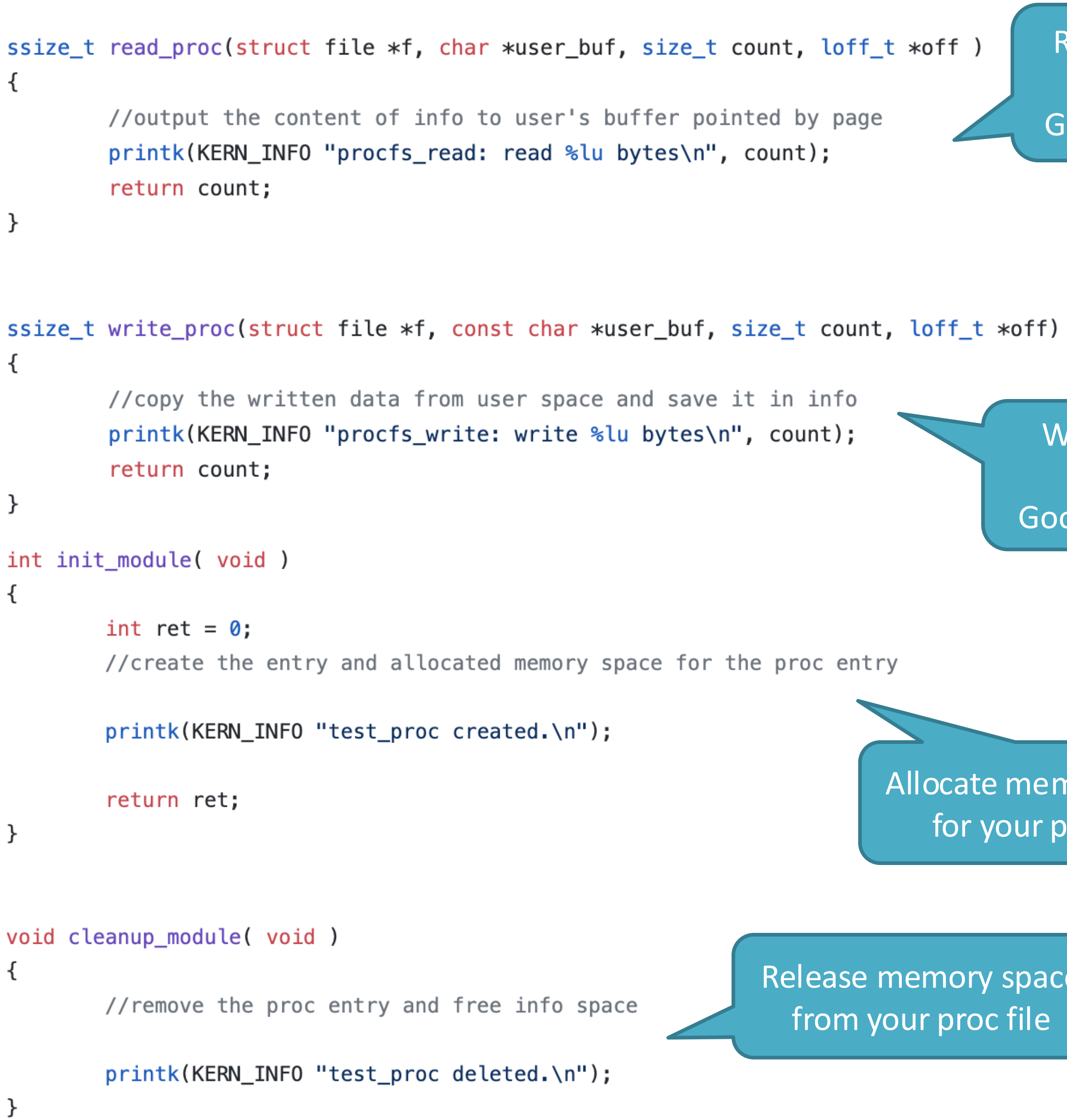
Part 2: after you remove your module, check whether it exists under /proc

- When your module is removed, it should disappear from the /proc



```
ksuo@ksuo-VirtualBox ~/hw4-2> sudo rmmod my_proc
fish: “sudo rmmod my_proc” terminated by signal SIGKILL (Forced quit)
ksuo@ksuo-VirtualBox ~/hw4-2> ls /proc/
1/     1229/  1314/  1460/  17/    275/ 38/  47/  52/  634/ 950/        fs/         mounts@       tty/
10/    1233/  1315/  1471/  173/   276/ 384/ 48/  523/ 659/ 954/        interrupts  mtrr          uptime
11/    1241/  1321/  15/    1791/  28/  39/  482/ 53/  677/ 975/        iomem       net@          version
1114/  1245/  1323/  1502/  18/    282/ 4/   484/ 532/ 683/ acpi/       ioports     pagetypeinfo  vmallocinfo
1119/  1252/  1325/  1504/  19/    29/  40/  485/ 533/ 686/ asound/     irq/        partitions    vmstat
1124/  1261/  1327/  1522/  192/   292/ 41/  489/ 534/ 7/   buddyinfo   kallsyms    pressure/     zoneinfo
1137/  1266/  1331/  154/   193/   3/   42/  49/  537/ 702/ bus/        kcore       sched_debug
1142/  1271/  1332/  155/   2/     30/  423/ 490/ 54/  748/ cgroups     keys        schedstat
1144/  1275/  1337/  156/   20/    32/  43/  491/ 56/  8/   cmdline     key-users   scsi/
1168/  1279/  1338/  157/   21/    321/ 438/ 493/ 571/ 803/ consoles    kmsg        self@
1174/  1283/  1372/  158/   22/    33/  44/  494/ 572/ 804/ cpuinfo     kpagecgroup slabinfo
1189/  1284/  1383/  159/   23/    331/ 440/ 496/ 59/  817/ crypto      kpagecount  softirqs
1190/  1285/  1384/  1598/  24/    335/ 449/ 497/ 6/   821/ devices     kpageflags  stat
12/    1287/  14/    16/    249/   336/ 45/  50/  60/  823/ diskstats   loadavg     swaps
1205/  1288/  1412/  161/   250/   34/  453/ 500/ 606/ 891/ dma         locks       sys/
1209/  1295/  1415/  162/   251/   35/  454/ 502/ 607/ 9/   driver/     mdstat      sysrq-trigger
1210/  13/    1439/  1684/  254/   36/  455/ 503/ 608/ 905/ execdomains meminfo     sysvipc/
1212/  1303/  1442/  1685/  26/    360/ 457/ 506/ 61/  912/ fb          misc        thread-self@
1218/  1304/  1446/  1695/  27/    37/  46/  509/ 614/ 931/ filesystems modules     timer_list
```

Part 2: read/write the proc entry your created in your module



```
ssize_t read_proc(struct file *f, char *user_buf, size_t count, loff_t *off )
{
        //output the content of info to user's buffer pointed by page
        printk(KERN_INFO "procfs_read: read %lu bytes\n", count);
        return count;
}


ssize_t write_proc(struct file *f, const char *user_buf, size_t count, loff_t *off)
{
        //copy the written data from user space and save it in info
        printk(KERN_INFO "procfs_write: write %lu bytes\n", count);
        return count;
}

int init_module( void )
{
        int ret = 0;
        //create the entry and allocated memory space for the proc entry

        printk(KERN_INFO "test_proc created.\n");

        return ret;
}


void cleanup_module( void )
{
        //remove the proc entry and free info space

        printk(KERN_INFO "test_proc deleted.\n");
}
```

Read data to user space from your proc file memory
Google search “copy_to_user”

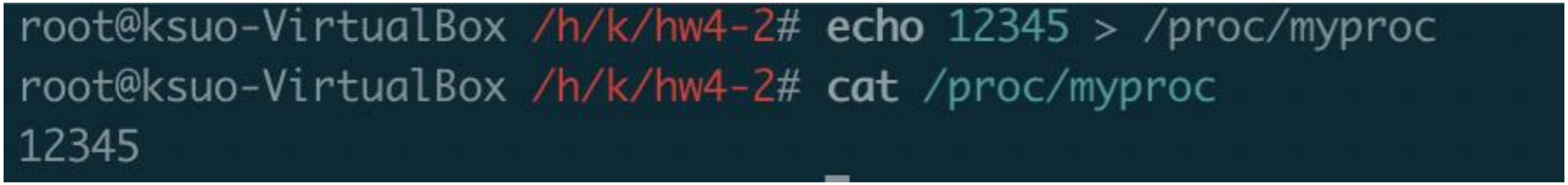
Write data from user space to your proc file memory
Google search “copy_from_user”

Allocate memory space for your proc file

Release memory space from your proc file

Part 2: read/write the proc entry your created in your module

- Use the following to test the read or write on the entry of proc file system
  - ○ # echo to write data into your proc entry
  - ○ # cat command to read data from your proc entry

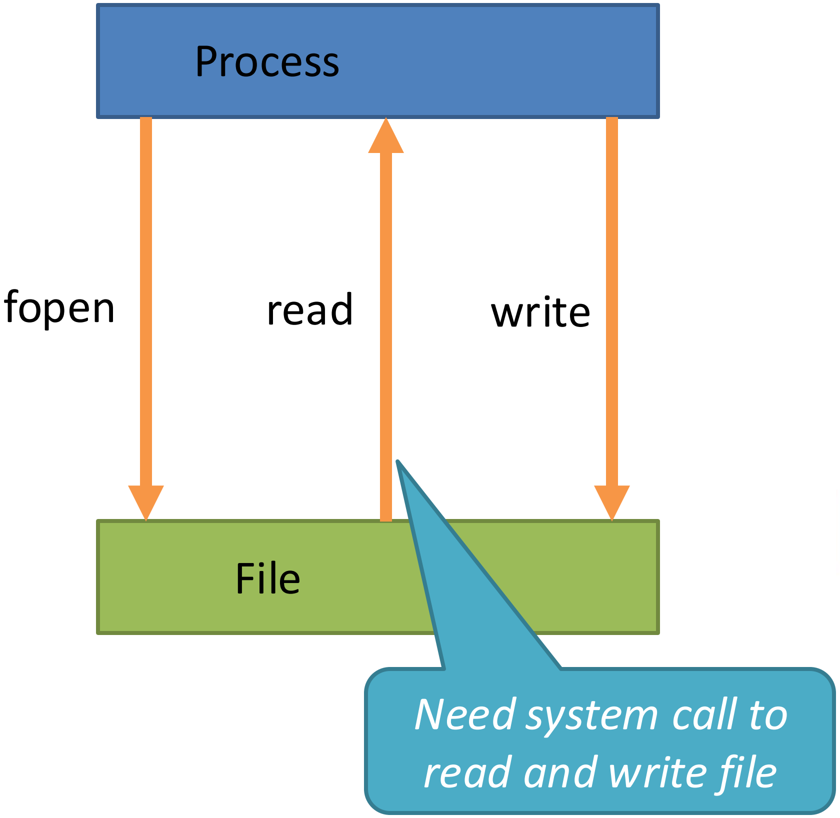


```
root@ksuo-VirtualBox /h/k/hw4-2# echo 12345 > /proc/myproc
root@ksuo-VirtualBox /h/k/hw4-2# cat /proc/myproc
12345
```

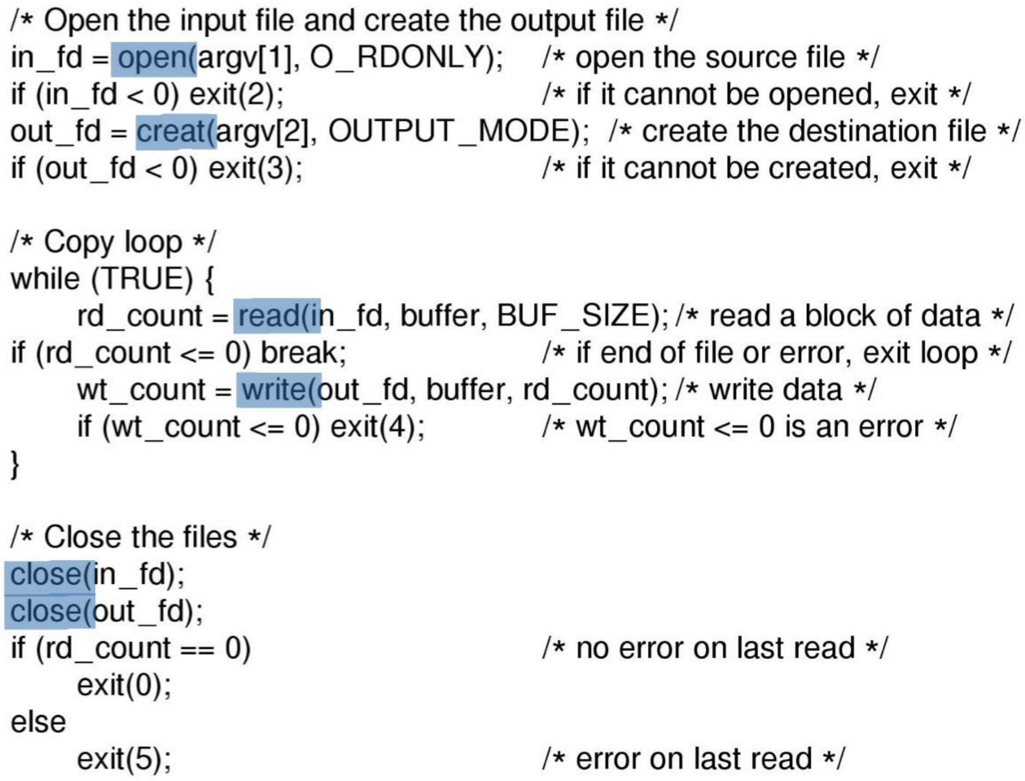
Outline

- Part 1: Create a helloworld kernel module (20’)
- Part 2: Create an entry in the /proc file system for user-level read and write (30’)
- Part 3: Exchange data between the user and kernel space via mmap (50’)

Memory-mapped Files

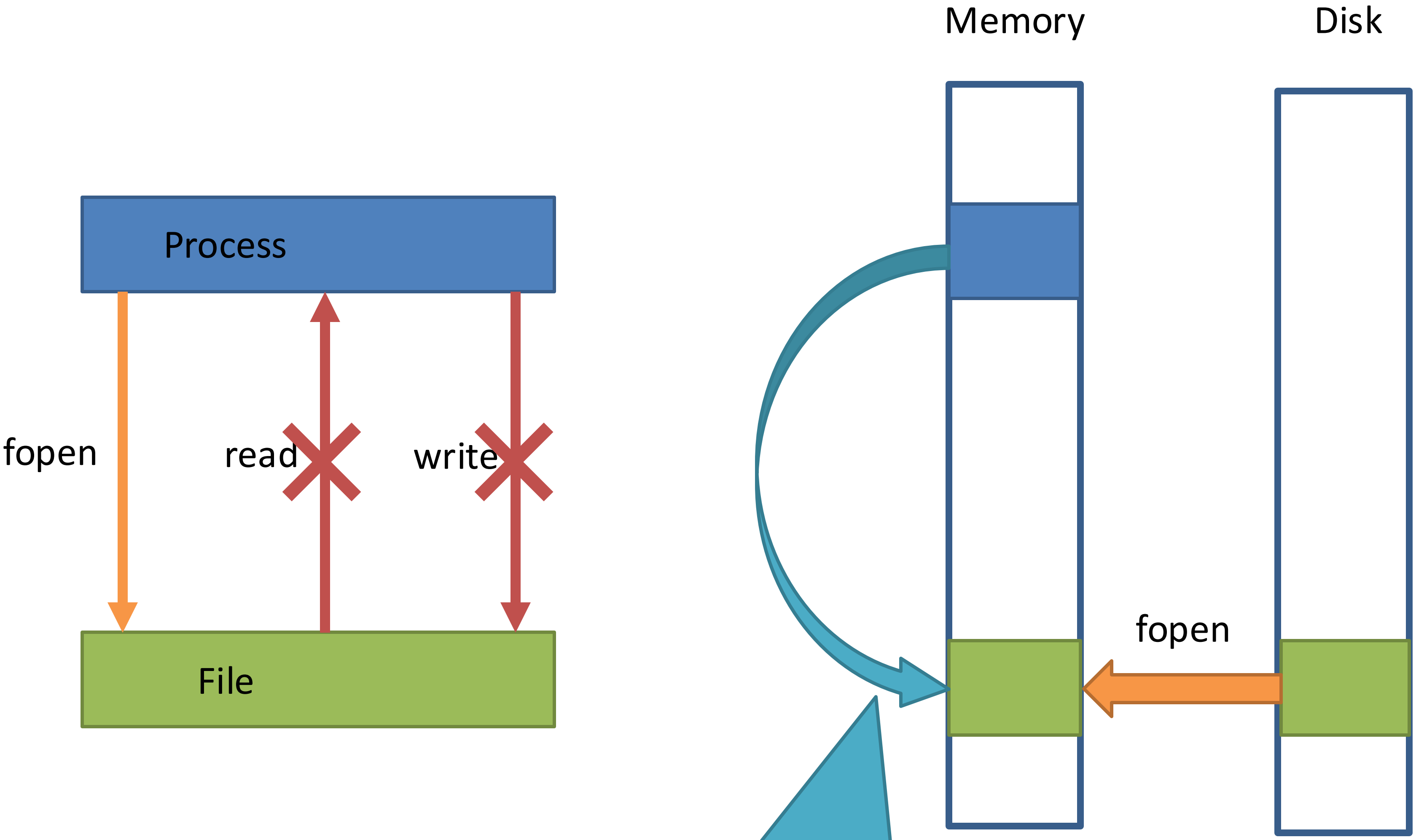


Need system call to read and write file



```
/* Open the input file and create the output file */
in_fd = open(argv[1], O_RDONLY);     /* open the source file */
if (in_fd < 0) exit(2);              /* if it cannot be opened, exit */
out_fd = creat(argv[2], OUTPUT_MODE);  /* create the destination file */
if (out_fd < 0) exit(3);             /* if it cannot be created, exit */

/* Copy loop */
while (TRUE) {
     rd_count = read(in_fd, buffer, BUF_SIZE); /* read a block of data */
if (rd_count <= 0) break;            /* if end of file or error, exit loop */
     wt_count = write(out_fd, buffer, rd_count); /* write data */
     if (wt_count <= 0) exit(4);     /* wt_count <= 0 is an error */
}

/* Close the files */
close(in_fd);
close(out_fd);
if (rd_count == 0)                   /* no error on last read */
     exit(0);
else
     exit(5);                        /* error on last read */
```

Memory-mapped Files

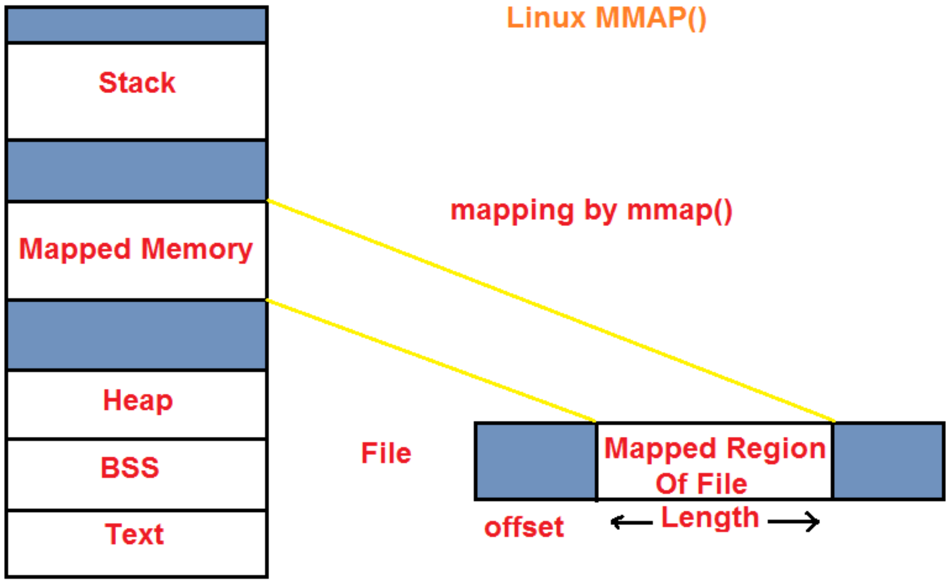


The file is in memory. Can we visit the address directly.

Memory-mapped Files

- OS provide a way (map and unmap) to map files into the address space of a running process
  - No read or write system calls are needed thereafter

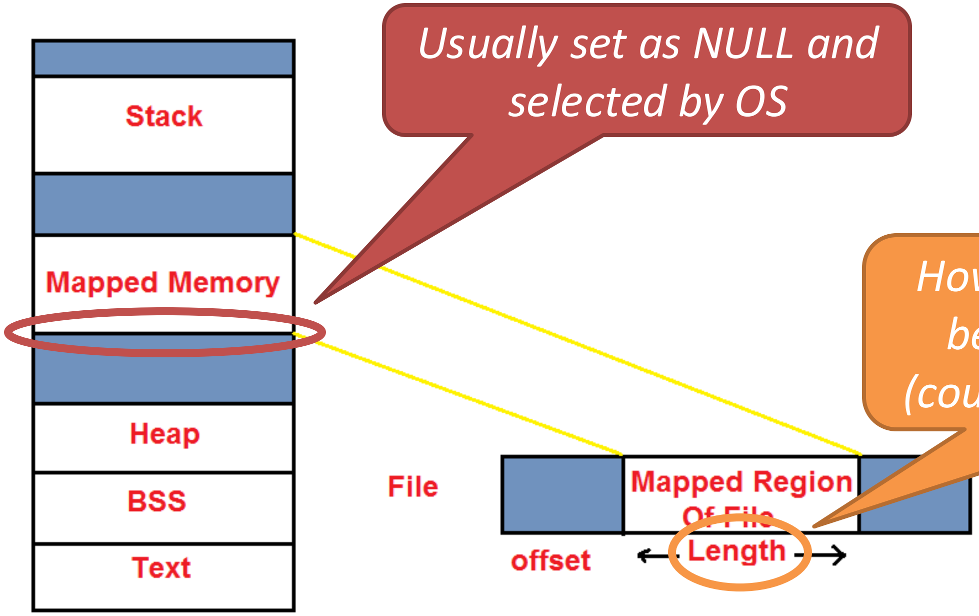
- Advantages
  - Improved I/O performance and avoidance of kernel to user data copying



Memory-mapped Files

```
#include <sys/mman.h>

void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);
```

https://pubs.opengroup.org/onlinepubs/009695399/functions/mmap.html



Usually set as NULL and selected by OS

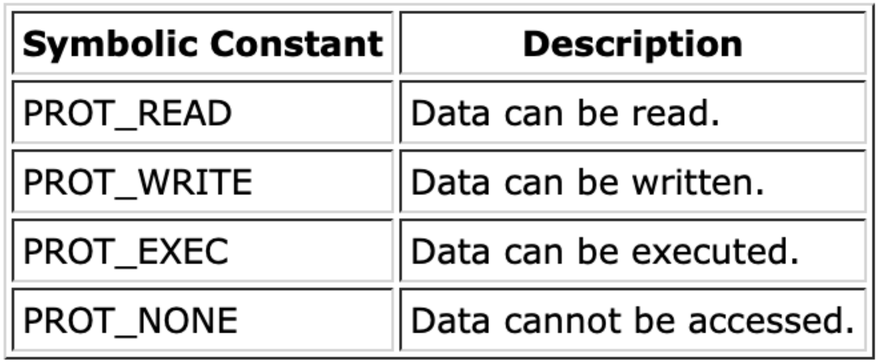
Start address

How much data in file will be mapped to memory (could be less than file size)

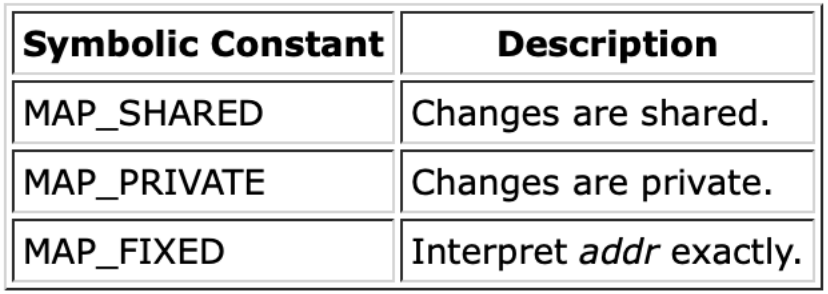
Memory-mapped Files

```
#include <sys/mman.h>

void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);
```

https://pubs.opengroup.org/onlinepubs/009695399/functions/mmap.html



Symbolic Constant	Description
PROT_READ	Data can be read.
PROT_WRITE	Data can be written.
PROT_EXEC	Data can be executed.
PROT_NONE	Data cannot be accessed.

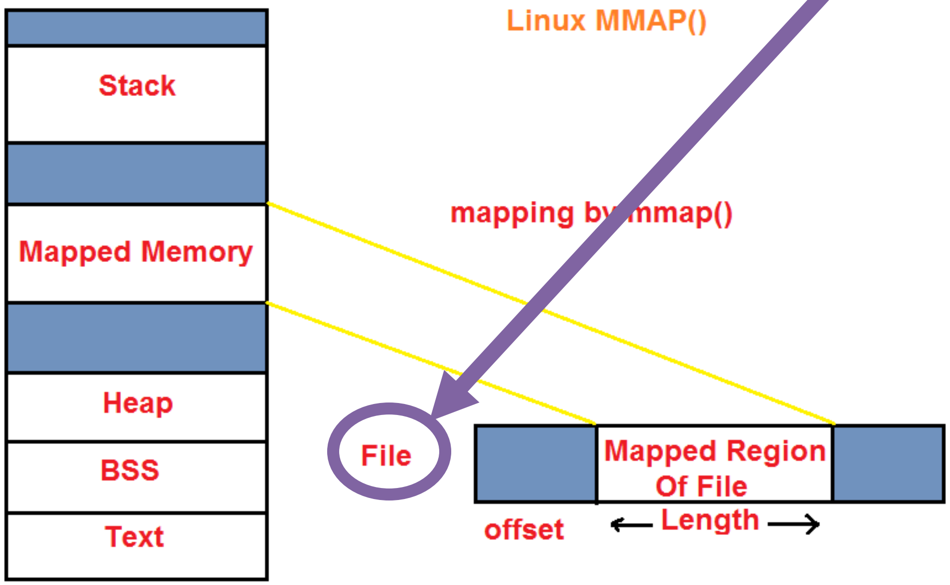


Symbolic Constant	Description
MAP_SHARED	Changes are shared.
MAP_PRIVATE	Changes are private.
MAP_FIXED	Interpret *addr* exactly.

Memory-mapped Files

```
#include <sys/mman.h>

void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);
```

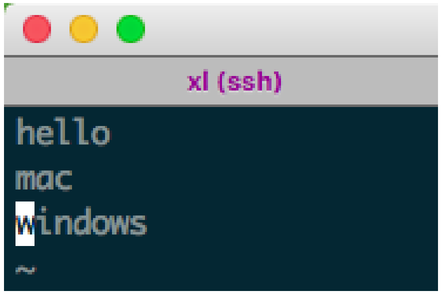
https://pubs.opengroup.org/onlinepubs/009695399/functions/mmap.html



Memory-mapped File Example

```
#include <sys/mman.h> /* for mmap and munmap */
#include <sys/types.h> /* for open */
#include <sys/stat.h> /* for open */
#include <fcntl.h> /* for open */
#include <unistd.h> /* for lseek and write */
#include <stdio.h>

int main(int argc, char **argv)
{
            int fd;
            char *mapped_mem, * p;
            int flength = 1024;
            void * start_addr = 0;

            fd = open(argv[1], O_RDWR | O_CREAT, S_IRUSR      IWUSR);
            flength = lseek(fd, 1, SEEK_END);
            lseek(fd, 0, SEEK_SET);

            mapped_mem = mmap(start_addr, flength, PROT_READ, MAP_PRIVATE, fd, 0);

            printf("%s\n", mapped_mem);
            close(fd);
            munmap(mapped_mem, flength);
            return 0;

}
```

text.txt



```
hello
mac
windows
~
```

Return the mapped memory address

Allow read

Set private, do not allow other process to read

Print out the data in the memory

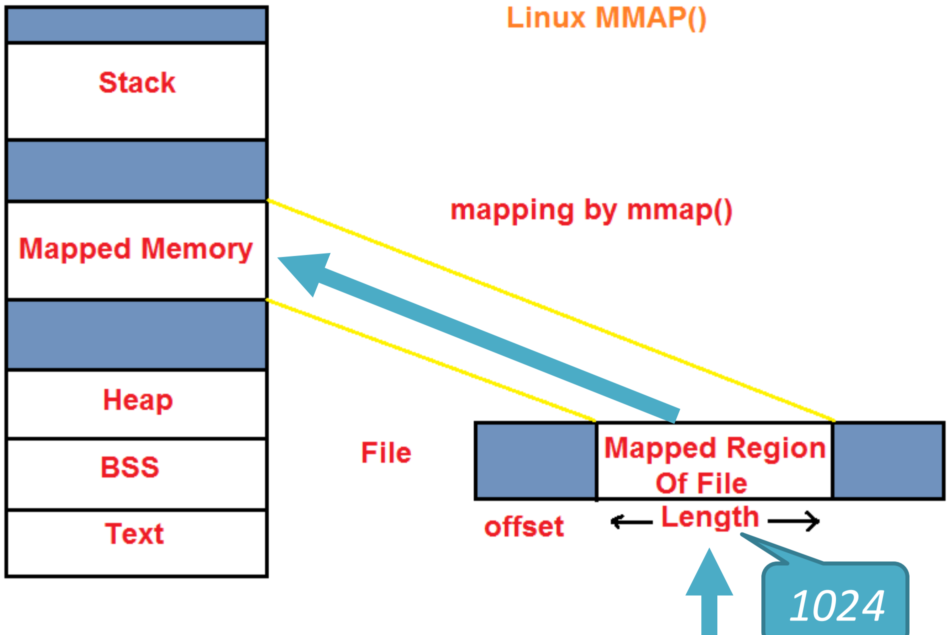
0: beginning of the file
1024: mapped memory size

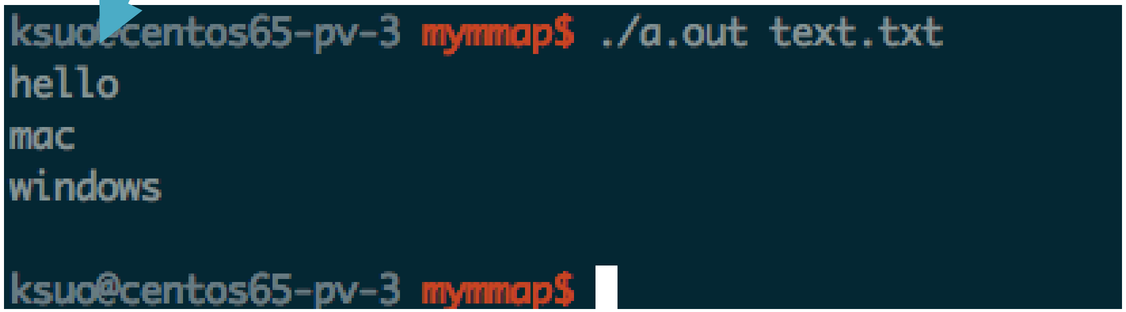
Memory-mapped File Example

```
#include <sys/mman.h> /* for mmap and munmap */
#include <sys/types.h> /* for open */
#include <sys/stat.h> /* for open */
#include <fcntl.h> /* for open */
#include <unistd.h> /* for lseek and write */
#include <stdio.h>

int main(int argc, char **argv)
{
            int fd;
            char *mapped_mem, * p;
            int flength = 1024;
            void * start_addr = 0;

            fd = open(argv[1], O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
            flength = lseek(fd, 1, SEEK_END);
            lseek(fd, 0, SEEK_SET);

            mapped_mem = mmap(start_addr, flength, PROT_READ, MAP_PRIVATE, fd, 0);

            printf("%s\n", mapped_mem);
            close(fd);
            munmap(mapped_mem, flength);
            return 0;
}
```

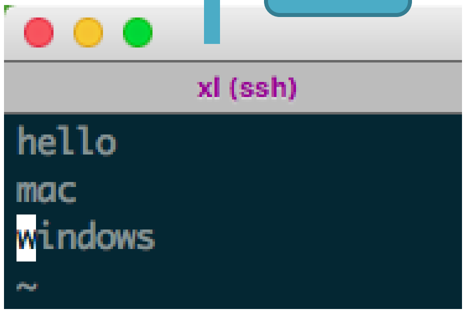
Memory-mapped File Example

Print out the data in this memory area





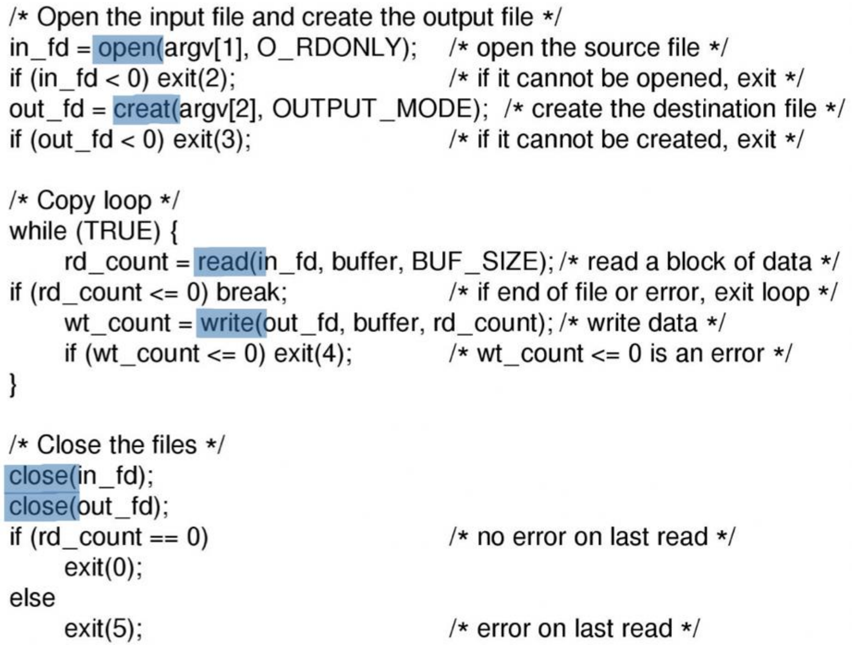
```
ksuo@centos65-pv-3 mymmap$ ./a.out text.txt
hello
mac
windows

ksuo@centos65-pv-3 mymmap$
```



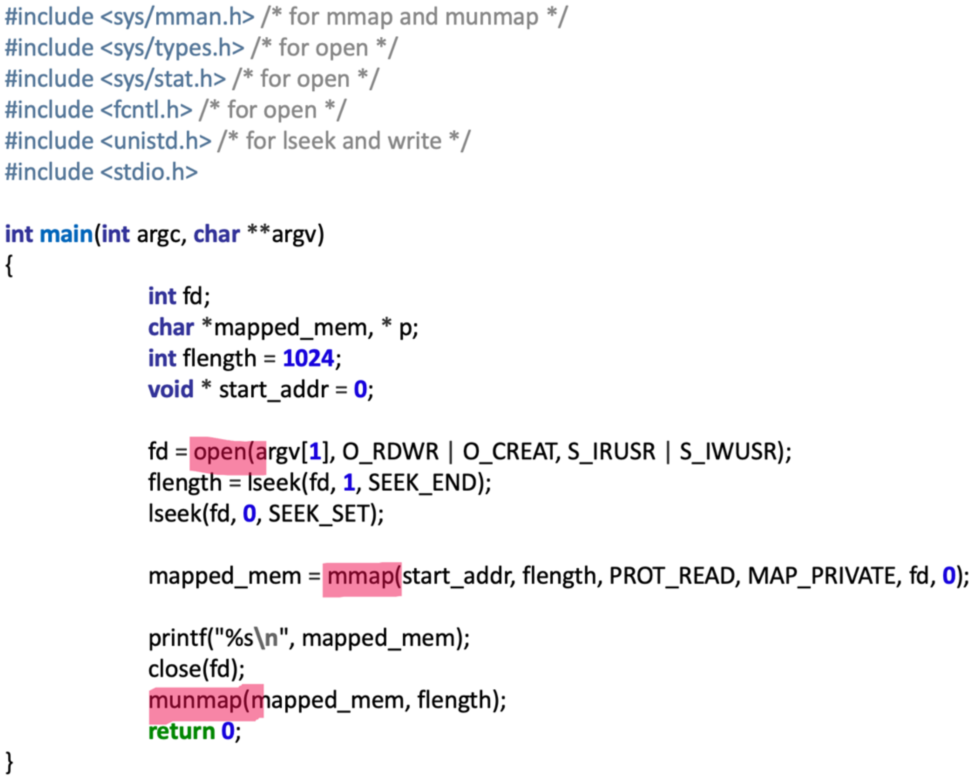
```
hello
mac
windows
~
```

text.txt

Comparison of regular file and memory-mapped file



```
/* Open the input file and create the output file */
in_fd = open(argv[1], O_RDONLY);        /* open the source file */
if (in_fd < 0) exit(2);                 /* if it cannot be opened, exit */
out_fd = creat(argv[2], OUTPUT_MODE);   /* create the destination file */
if (out_fd < 0) exit(3);                /* if it cannot be created, exit */

/* Copy loop */
while (TRUE) {
     rd_count = read(in_fd, buffer, BUF_SIZE); /* read a block of data */
if (rd_count <= 0) break;               /* if end of file or error, exit loop */
     wt_count = write(out_fd, buffer, rd_count); /* write data */
     if (wt_count <= 0) exit(4);        /* wt_count <= 0 is an error */
}

/* Close the files */
close(in_fd);
close(out_fd);
if (rd_count == 0)                      /* no error on last read */
     exit(0);
else
     exit(5);                           /* error on last read */
```

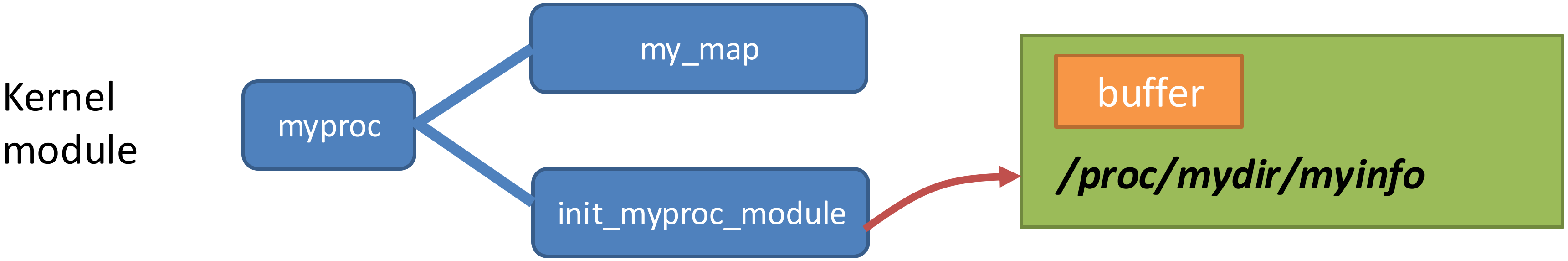


```
#include <sys/mman.h> /* for mmap and munmap */
#include <sys/types.h> /* for open */
#include <sys/stat.h> /* for open */
#include <fcntl.h> /* for open */
#include <unistd.h> /* for lseek and write */
#include <stdio.h>

int main(int argc, char **argv)
{
        int fd;
        char *mapped_mem, * p;
        int flength = 1024;
        void * start_addr = 0;

        fd = open(argv[1], O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
        flength = lseek(fd, 1, SEEK_END);
        lseek(fd, 0, SEEK_SET);

        mapped_mem = mmap(start_addr, flength, PROT_READ, MAP_PRIVATE, fd, 0);

        printf("%s\n", mapped_mem);
        close(fd);
        munmap(mapped_mem, flength);
        return 0;
}
```

Part 3: Exchange data between the user and kernel space via mmap

- https://github.com/kevinsuo/CS3502/blob/master/project-4-3-1.c

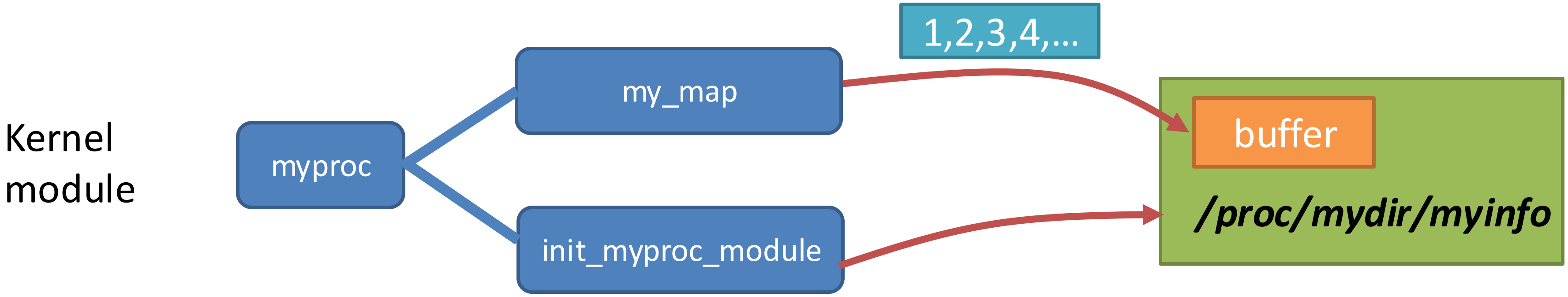
- The above code will create an entry ***/proc/mydir/myinfo*** under the proc file system and allocate a buffer under this entry



Part 3: Exchange data between the user and kernel space via mmap

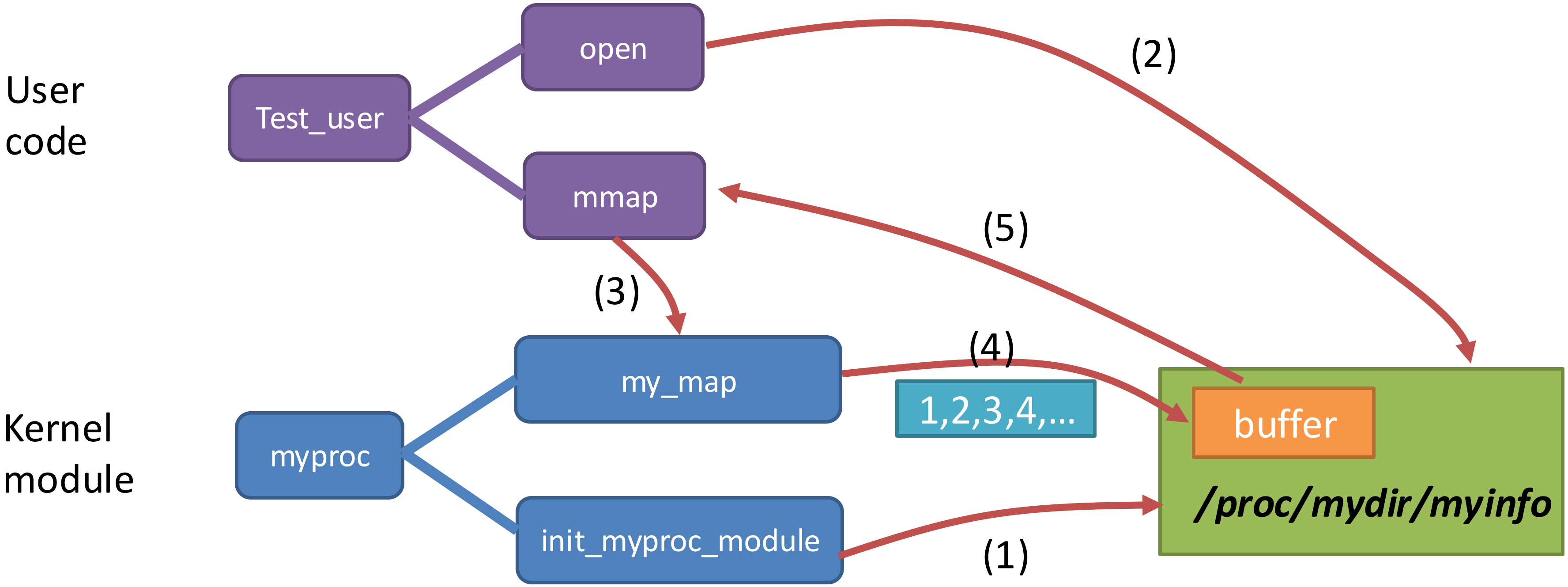
- You are required to implement the ***my_map*** function to map one piece of memory (***char array[12]***) into user space.

```
static unsigned char array[12]={0,1,2,3,4,5,6,7,8,9,10,11};
```



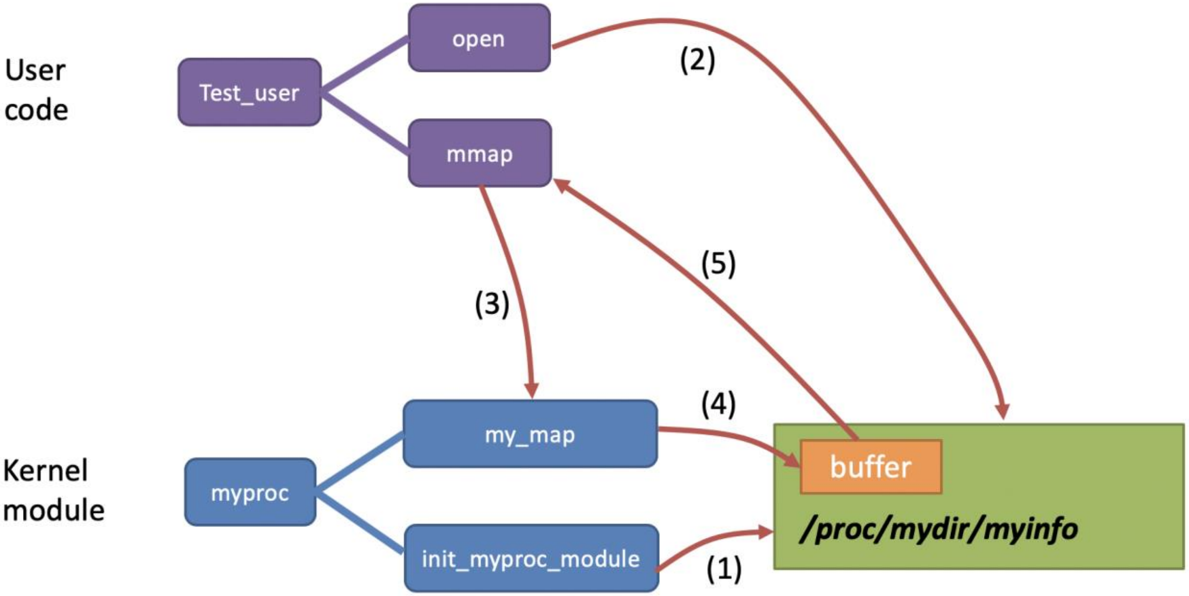
Part 3: Exchange data between the user and kernel space via mmap

- You are required to write a user space program using mmap to visit the memory space of the proc file and print the data in that memory area.
- https://github.com/kevinsuo/CS3502/blob/master/project-4-3-2.c



Part 3: Exchange data between the user and kernel space via mmap

1. Kernel module create a proc file: ***/proc/mydir/myinfo***
2. User process open the created proc file
3. User process calls mmap function, which further executed my_map defined in the kernel
4. my_map() then maps one piece of memory into user space (e.g., buffer) and puts some data inside
5. User process visits this piece of memory and prints the data out.



Conclusion

- Part 1: Create a helloworld kernel module (20’)
- Part 2: Create an entry in the /proc file system for user level read and write (30’)
- Part 3: Exchange data between the user and kernel space via mmap (50’)