

CS 3502

Operating Systems

Project 1

Kun Suo

Computer Science, Kennesaw State University

<https://kevinsuo.github.io/>

Outline

- **Assignment 0: create a VM and compile your kernel**
- Assignment 1: add a new system call into the Linux kernel
- Assignment 2: Extend your new system call to print out the calling process's information
- Assignment 3: Extend your new system call to print out the information of an arbitrary process identified by its PID



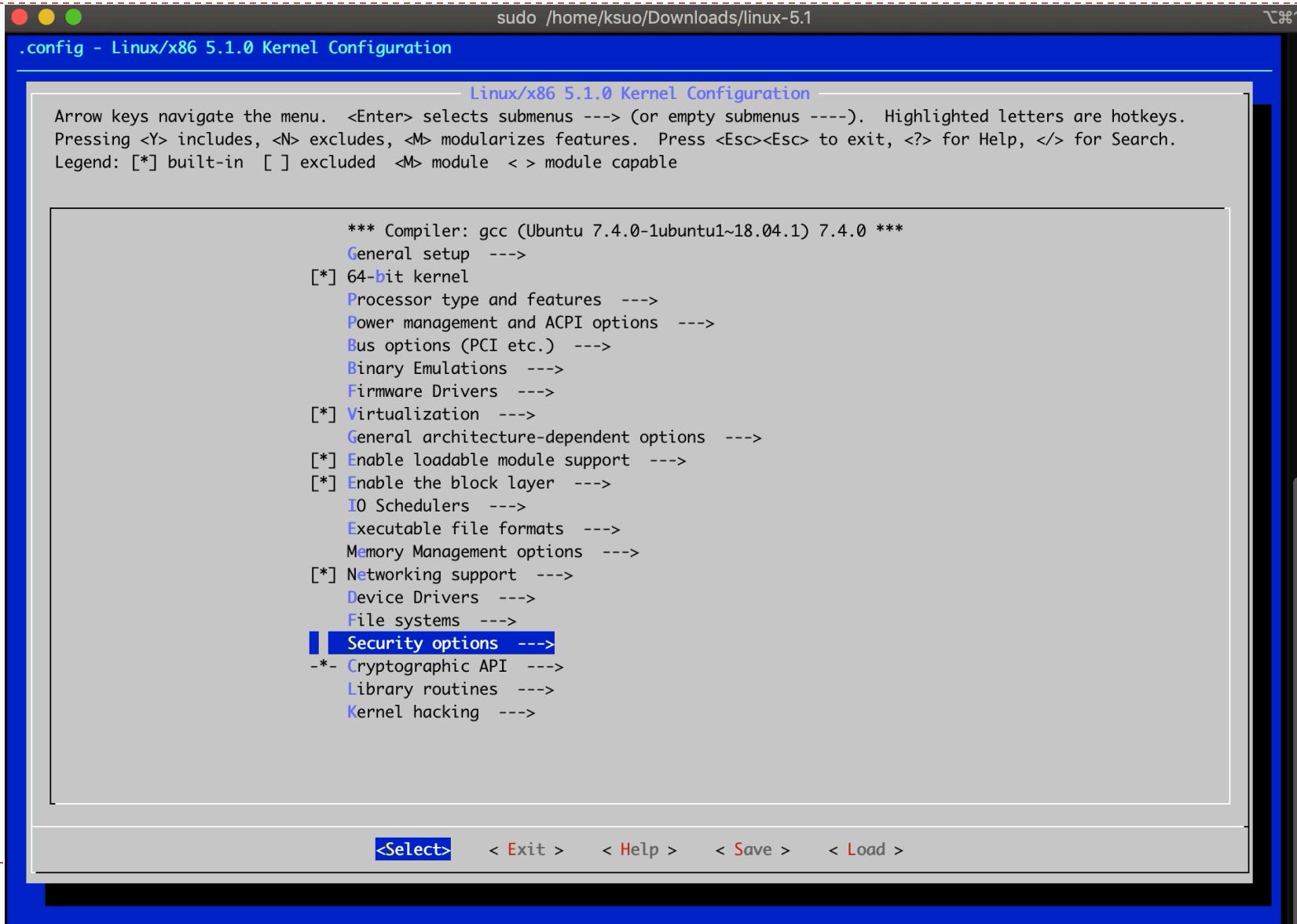
How to build one ubuntu VM?

- HostOS
 - Windows 10:
<https://www.youtube.com/watch?v=QbmRXJJKsvs>
 - MacOS:
<https://www.youtube.com/watch?v=GDoCrfPma2k&t=321s>



Project 1: menuconfig

<https://youtu.be/UyOGF4UOoR0>



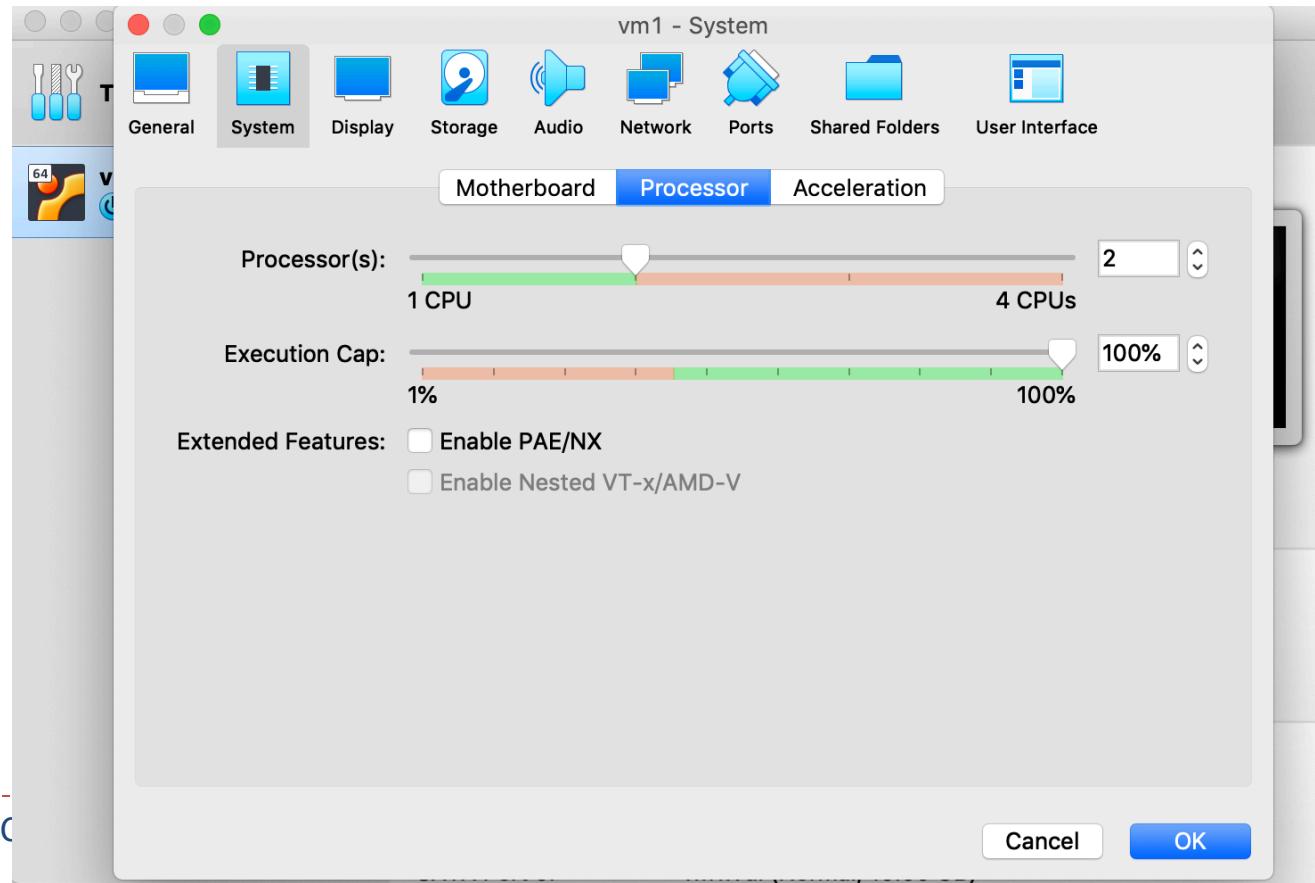
Tips

- Compile Linux kernel takes half to many hours, depending on machine speed
- To save the time, you can compile it before you sleep
- Run many commands in one:
\$ sudo make; sudo make modules; sudo make modules_install; sudo make install

```
ksuo@ksuo-VirtualBox ~/D/linux-5.1>
sudo make; sudo make modules; sudo make modules_install; sudo make install
```

Tips

- Set up more CPUs for VM
- Make `-j N`, to accelerate compiling



Where is my kernel?

- \$ ls /boot/

Initial ramdisk: loading a temporary root file system into memory. Used for startup.

```
ksuo@ksuo-VirtualBox: ~
```

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

vm [Running]
Thu 11:38

Linux executable kernel image

```
fish /home/ksuo/D/linux-5.1> ls /boot/
memtest86+.elf
memtest86+_multiboot.bin
System.map-5.0.0-23-generic
System.map-5.0.0-25-generic
System.map-5.1.0
vmlinuz-5.0.0-23-generic
vmlinuz-5.0.0-25-generic
vmlinuz-5.1.0
```



Which kernel to boot if there are many?

If you are using Ubuntu: change the grub configuration file:

```
$ sudo vim /etc/default/grub
```

The OS boots by using the first kernel by default. You have 10 seconds to choose.

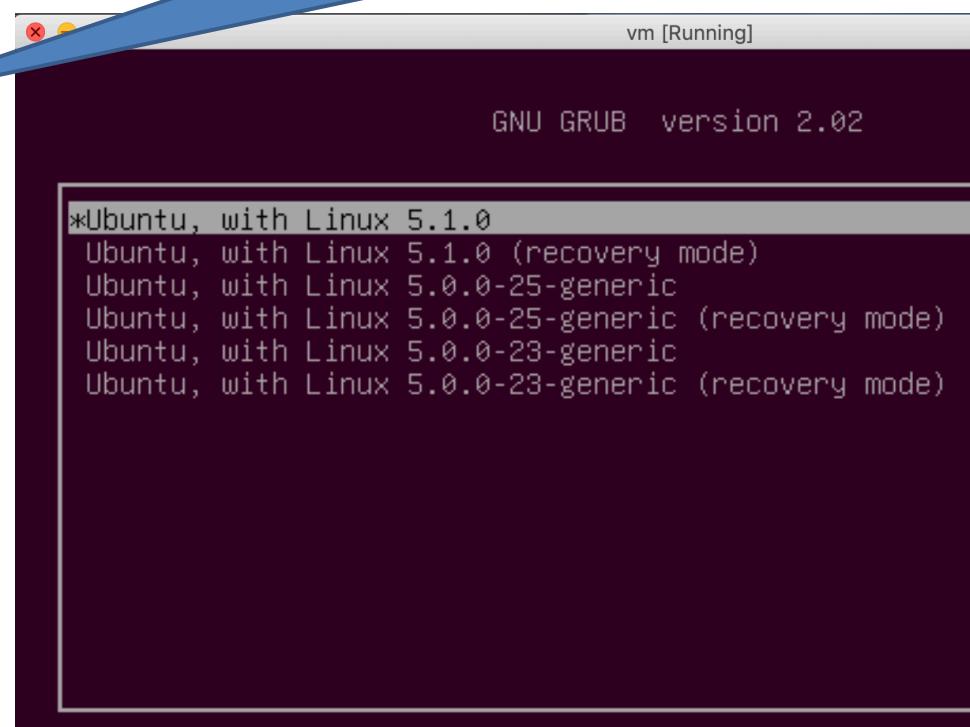
Make the following changes:

```
GRUB_DEFAULT=0
```

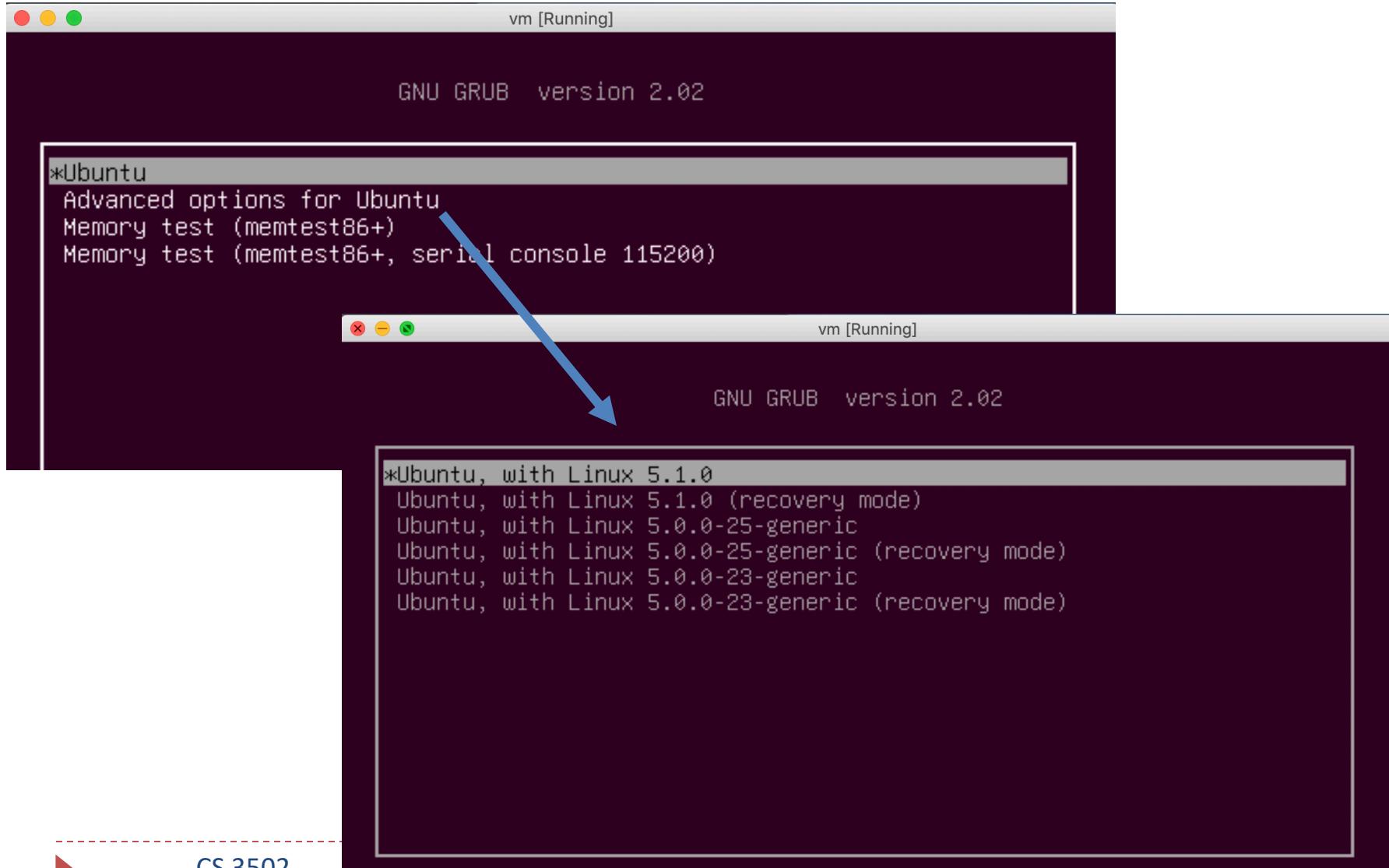
```
GRUB_TIMEOUT=10
```

Then, update the grub entry:

```
$ sudo update-grub2
```

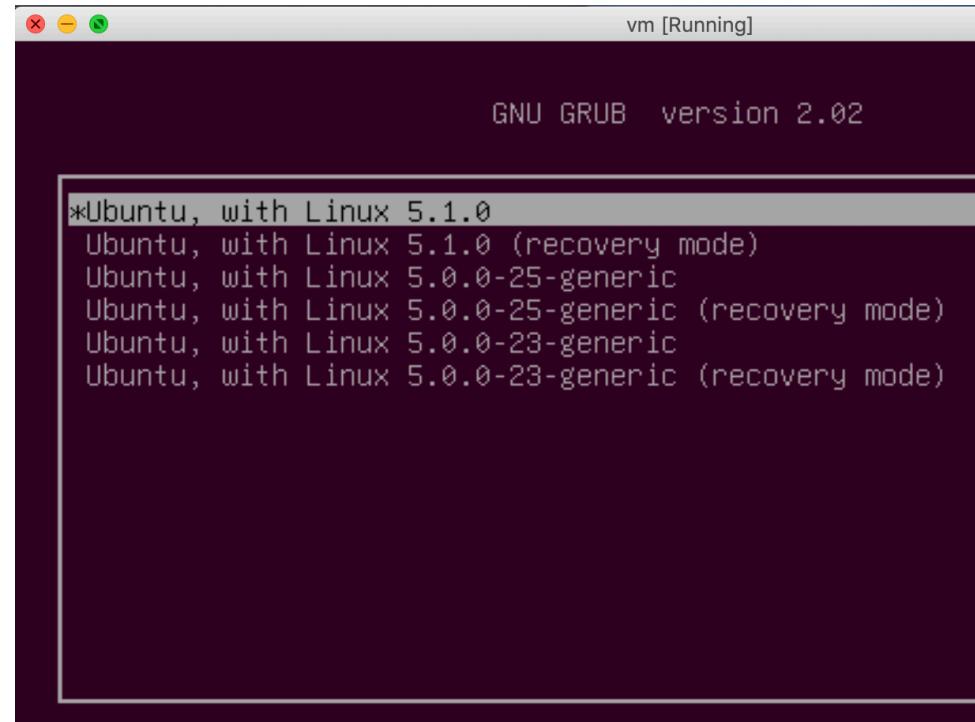


Which kernel to boot if there are many?



What if my kernel crashed?

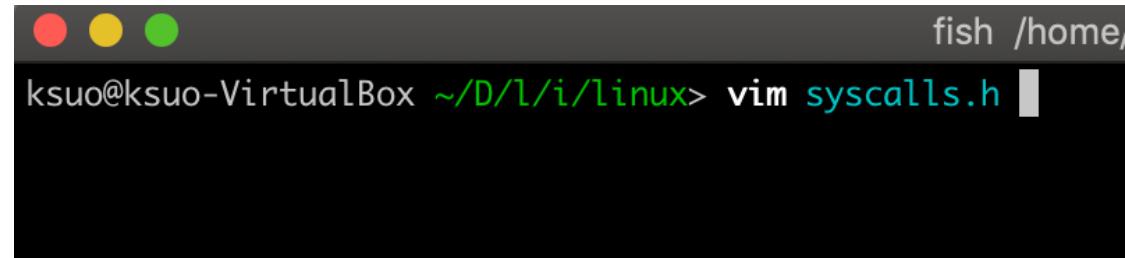
- Your kernel could crash because you might bring in some kernel bugs
- In the menu, choose the old kernel to boot the system
- Fix your bug in the source code
- Compile and reboot



Project 1: No IDE, please use vim

- Open a file

- \$ vim test.c



```
fish /home/ksuo@ksuo-VirtualBox ~/D/l/i/linux> vim syscalls.h
```

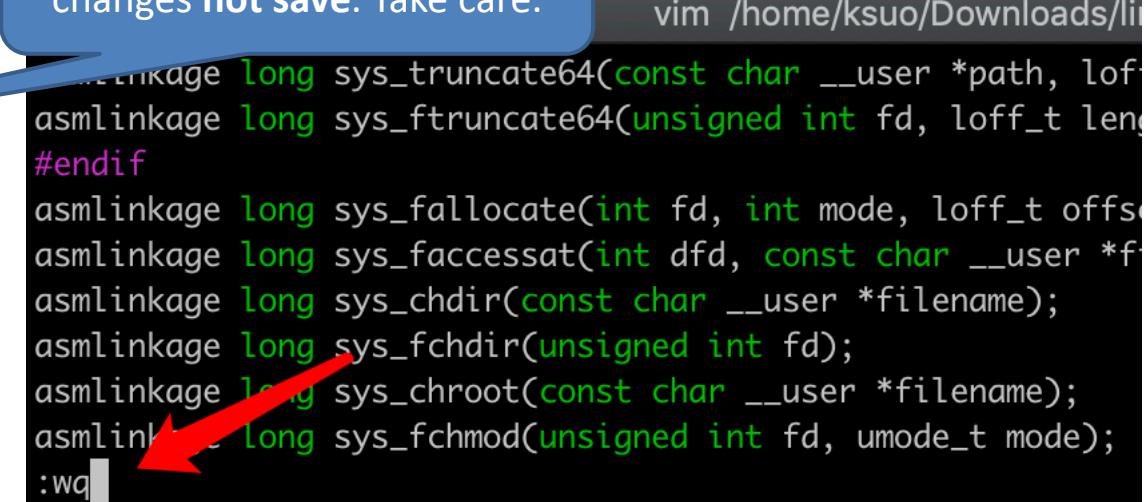
- Close a file

- Inside vim (after opening), press :q!

colon mark means menu.

Exclamation mark means changes **not save**. Take care.

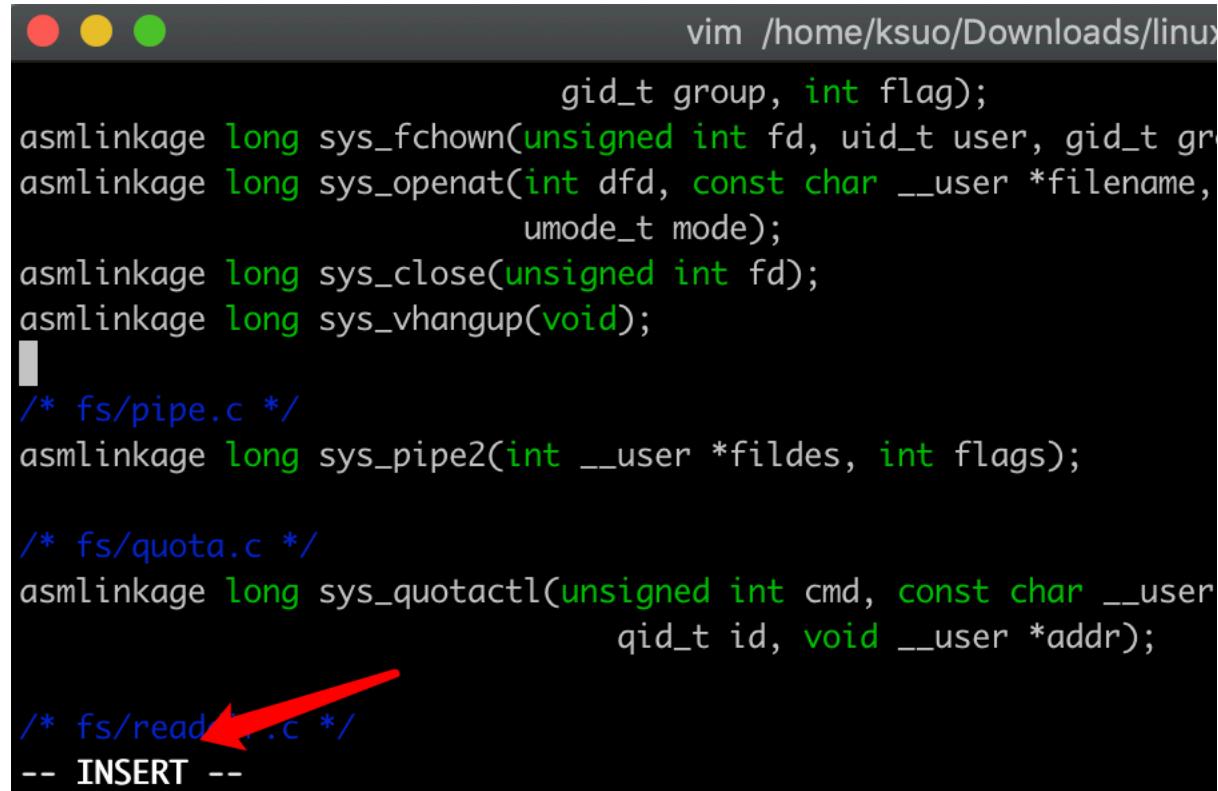
- Inside vim (after opening), press :wq, changes saved



```
asmlinkage long sys_truncate64(const char __user *path, loff_t len);  
asmlinkage long sys_ftruncate64(unsigned int fd, loff_t len);  
#endif  
asmlinkage long sys_fallocate(int fd, int mode, loff_t offset, loff_t len);  
asmlinkage long sys_faccessat(int dfd, const char __user *filename, int mode);  
asmlinkage long sys_chdir(const char __user *filename);  
asmlinkage long sys_fchdir(unsigned int fd);  
asmlinkage long sys_chroot(const char __user *filename);  
asmlinkage long sys_fchmod(unsigned int fd, umode_t mode);  
:wq
```

Project 1: No IDE, please use vim

- Edit a file
 - Open a file
 - Press i (means enter the insert mode). then input your code
 - Press ESC to exit the insert mode
 - Close a file (:wq)



```
vim /home/ksuo/Downloads/linux-headers-4.15.0-38/include/linux/fs.h
gid_t group, int flag);
asm linkage long sys_fchown(unsigned int fd, uid_t user, gid_t gr
asm linkage long sys_openat(int dfd, const char __user *filename,
                           umode_t mode);
asm linkage long sys_close(unsigned int fd);
asm linkage long sys_vhangup(void);

/*
 * fs/pipe.c */
asm linkage long sys_pipe2(int __user *fildes, int flags);

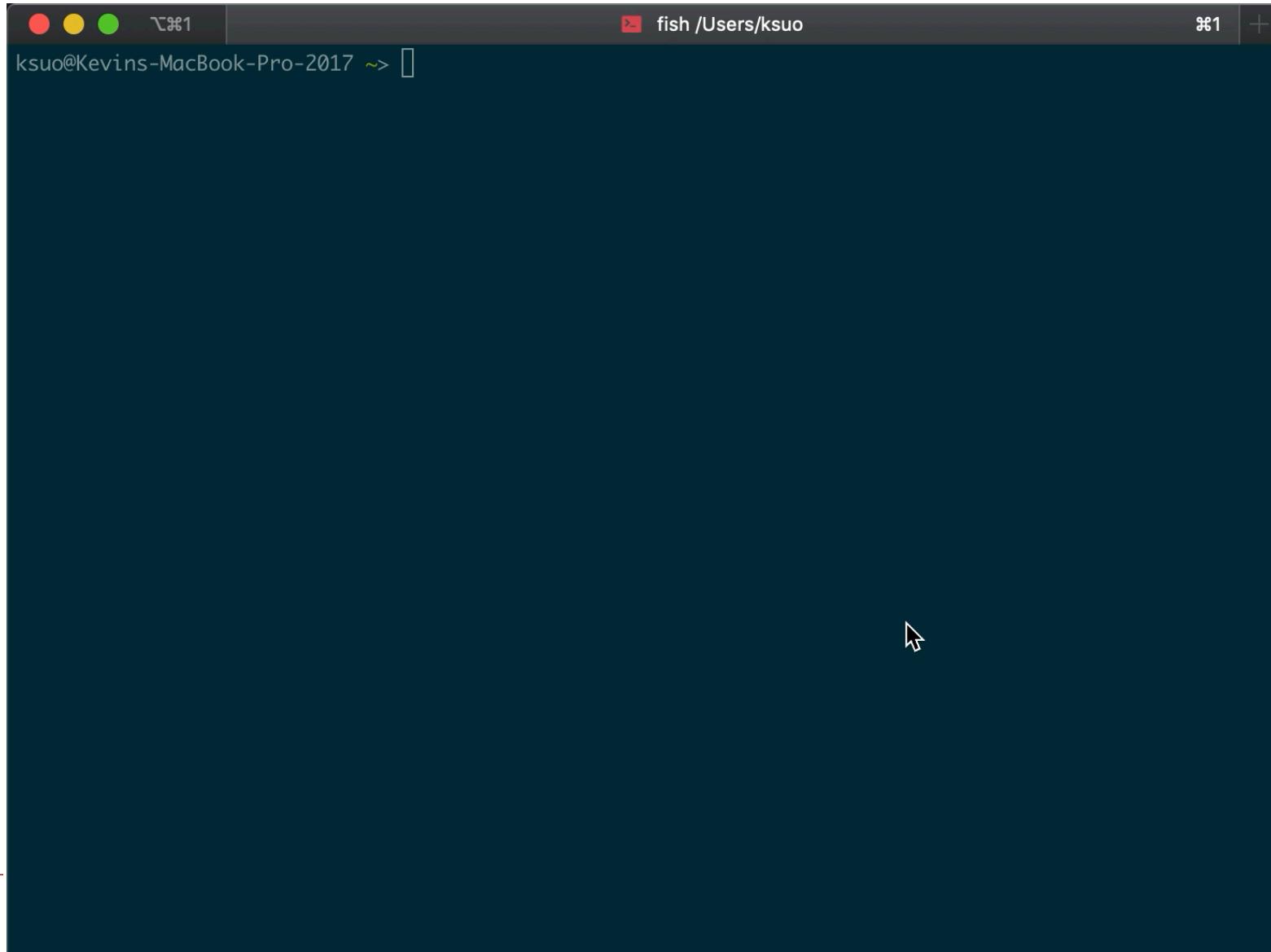
/*
 * fs/quota.c */
asm linkage long sys_quotactl(unsigned int cmd, const char __user
                               qid_t id, void __user *addr);

/*
 * fs/read.c */
-- INSERT --
```



Project 1: No IDE, please use vim

<https://youtu.be/rSOXZpLtVlo>



Edit a file with vim

- step 1: \$ vim file
- step 2: press **i**, enter insert mode; move the cursor to position and edit the context
- step 3: after editing, press **ESC** to exit the insert mode to normal mode
- step 4: press **:wq** to save what you edit and quit. If you do not want to save, press **:q!**



More about vim

- A quick start guide for beginners to the Vim text editor
 - <https://eastmanreference.com/a-quick-start-guide-for-beginners-to-the-vim-text-editor>
- Vim basics:
 - <https://www.howtoforge.com/vim-basics>
- Learn the Basic Vim Commands [Beginners Guide]
 - https://www.youtube.com/watch?time_continue=265&v=ZEGqkam-3lc



Compile the kernel

1. download the kernel source code
2. unzip the file
3. use *make menuconfig* to create a config file
4. use *make/make modules* to compile the code
5. use *make modules_install/make install* to
install the kernel
6. reboot the VM



Outline

- Assignment 0: create a VM and compile your kernel
- **Assignment 1: add a new system call into the Linux kernel**
- Assignment 2: Extend your new system call to print out the calling process's information
- Assignment 3: Extend your new system call to print out the information of an arbitrary process identified by its PID

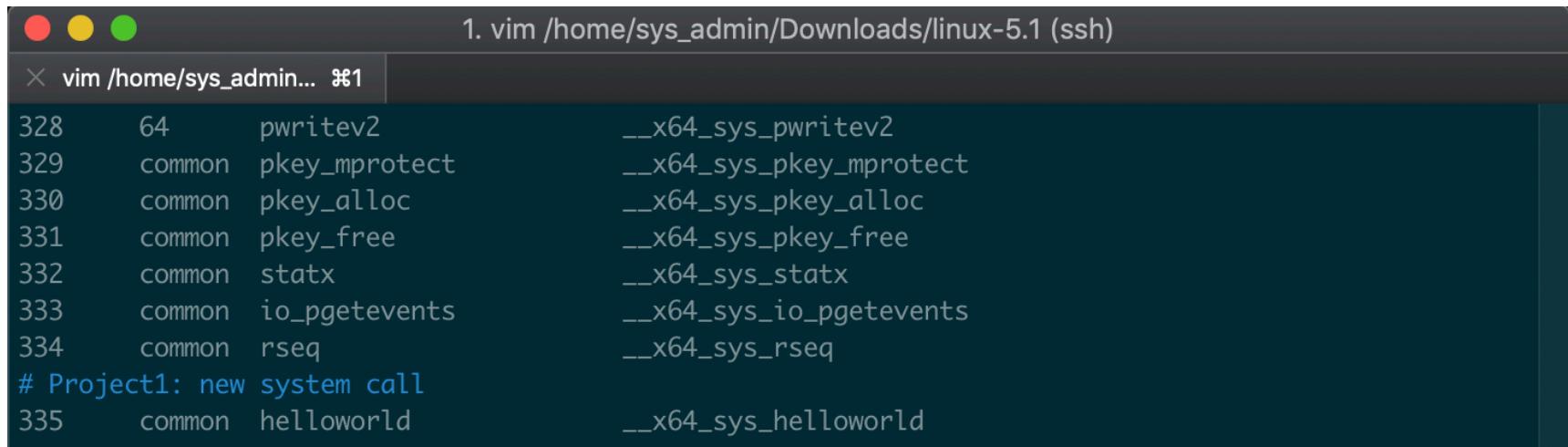


Define your system call

- Step 1: register your system call
- Step 2: declare your system call in the header file
- Step 3: implement your system call
- Step 4: write user level app to call it

Step 1: register your system call

arch/x86/entry/syscalls/syscall_64.tbl



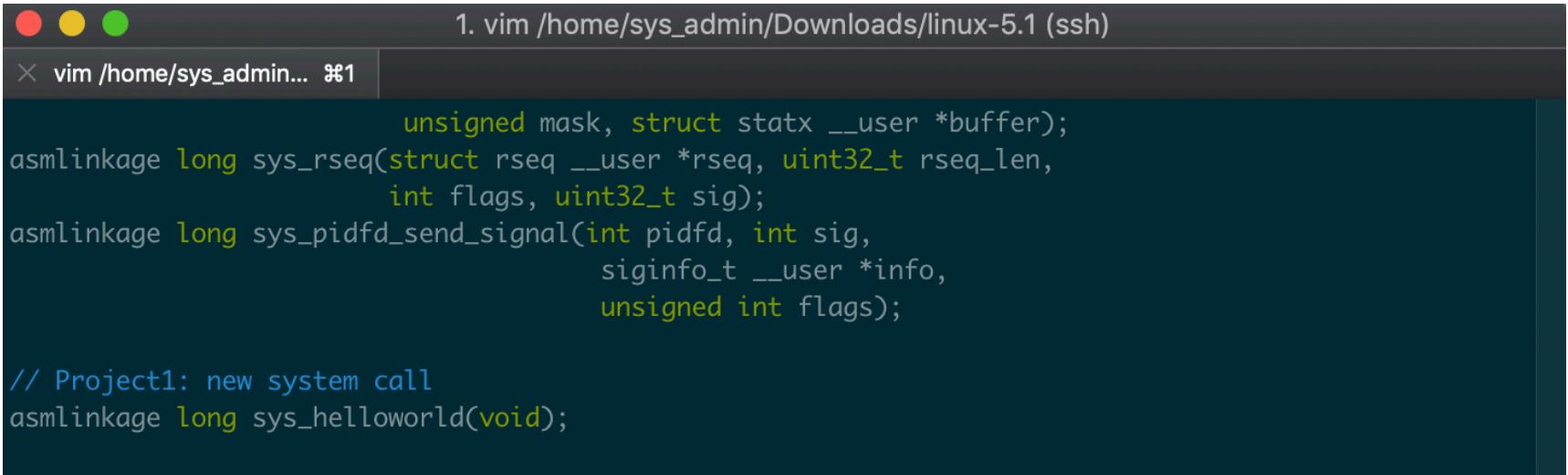
```
1. vim /home/sys_admin/Downloads/linux-5.1 (ssh)
vim /home/sys_admin... ⌘1
328      64      pwritev2          __x64_sys_pwritev2
329  common  pkey_mprotect     __x64_sys_pkey_mprotect
330  common  pkey_alloc        __x64_sys_pkey_alloc
331  common  pkey_free         __x64_sys_pkey_free
332  common  statx            __x64_sys_statx
333  common  io_pgetevents    __x64_sys_io_pgetevents
334  common  rseq              __x64_sys_rseq
# Project1: new system call
335  common  helloworld       __x64_sys_helloworld
```

https://elixir.bootlin.com/linux/v5.0/source/arch/x86/entry/syscalls/syscall_64.tbl#L346



Step 2: declare your system call in the header file

include/linux/syscalls.h



The screenshot shows a terminal window titled "1. vim /home/sys_admin/Downloads/linux-5.1 (ssh)". The vim editor is displaying the contents of the syscalls.h header file. The code includes several standard Linux system calls like sys_rseq and sys_pidfd_send_signal, followed by a new user-defined system call sys_helloworld.

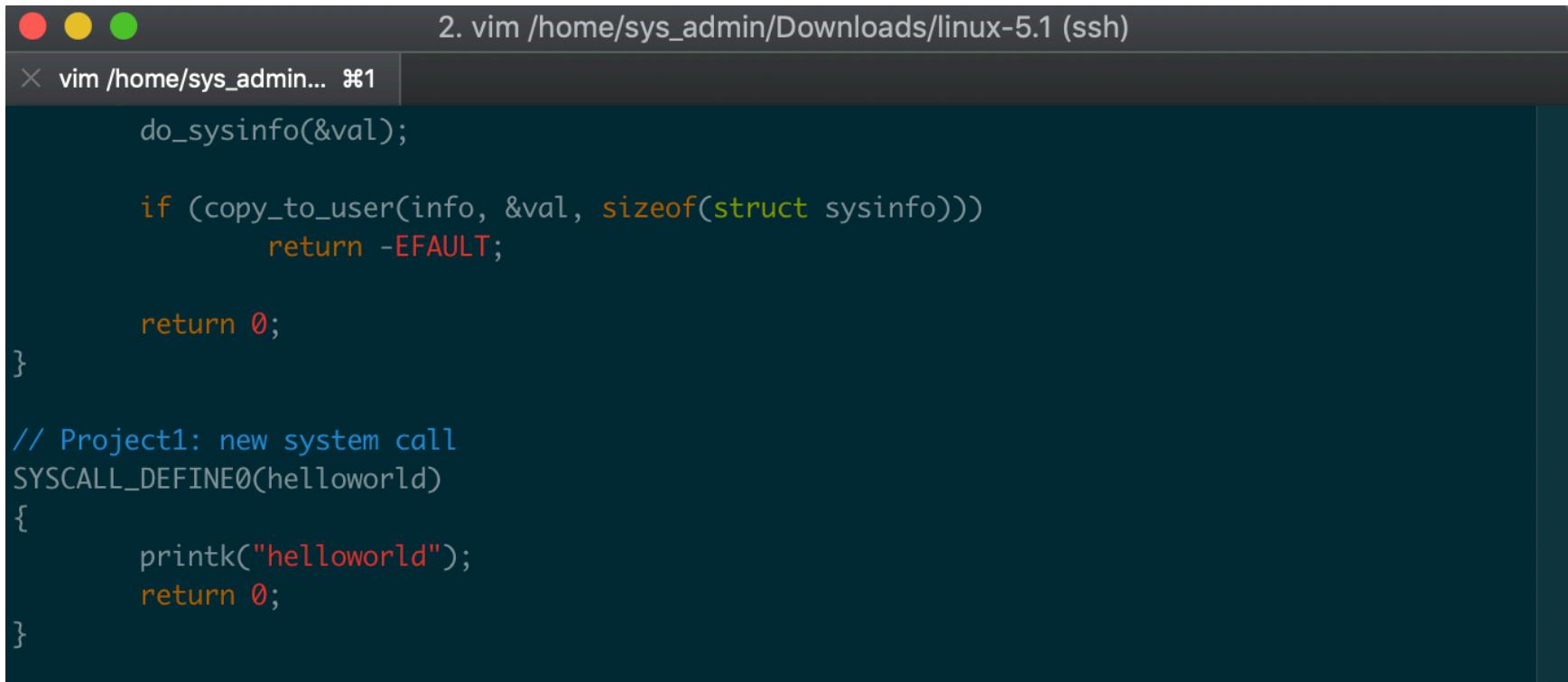
```
unsigned mask, struct statx __user *buffer);  
asmlinkage long sys_rseq(struct rseq __user *rseq, uint32_t rseq_len,  
                         int flags, uint32_t sig);  
asmlinkage long sys_pidfd_send_signal(int pidfd, int sig,  
                                       siginfo_t __user *info,  
                                       unsigned int flags);  
  
// Project1: new system call  
asmlinkage long sys_helloworld(void);
```

<https://elixir.bootlin.com/linux/v5.0/source/include/linux/syscalls.h>



Step 3: implement your system call

kernel/sys.c



```
2. vim /home/sys_admin/Downloads/linux-5.1 (ssh)
vim /home/sys_admin... #1

do_sysinfo(&val);

if (copy_to_user(info, &val, sizeof(struct sysinfo)))
    return -EFAULT;

return 0;
}

// Project1: new system call
SYSCALL_DEFINE0(helloworld)
{
    printk("helloworld");
    return 0;
}
```

<https://elixir.bootlin.com/linux/v5.0/source/kernel/sys.c#L402>



Step 4: write user level app to call it

test_syscall.c:

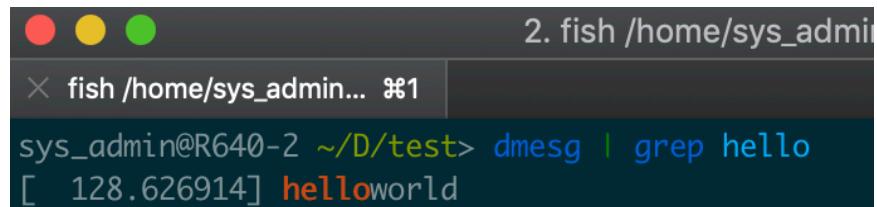
```
*****  
#include <linux/unistd .h>  
#include <sys/syscall .h>  
#include <sys/types .h>  
#include <stdio .h>  
#define __NR_helloworld 335  
  
int main(int argc, char *argv[])  
{  
    syscall (__NR_helloworld) ;  
    return 0 ;  
}  
*****
```

//If syscall needs parameter, then:
//syscall (__NR_helloworld, a, b, c) ;

Compile and execute:

```
$ gcc test_syscall.c -o test_syscall  
$ ./test_syscall
```

The test program will call the new system call and output a helloworld message at the tail of the output of dmesg (system log).



A screenshot of a terminal window titled "fish /home/sys_admin... #1". The command entered is "dmesg | grep hello". The output shows the message "[128.626914] helloworld" at the end of the log.

```
2. fish /home/sys_admin... #1  
x fish /home/sys_admin... #1  
sys_admin@R640-2 ~/D/test> dmesg | grep hello  
[ 128.626914] helloworld
```

Put it all together

User space

Step 4: user call it

test_syscall.c:

```
*****  
#include <linux/unistd.h>  
#include <sys/syscall.h>  
#include <sys/types.h>  
#include <stdio.h>  
#define __NR_helloworld 335  
  
int main(int argc, char *argv[]){  
    syscall (__NR_helloworld);  
    return 0;  
}  
*****
```

```
2. fish /home/sys_admin... #1  
x fish /home/sys_admin... #1  
sys_admin@R640-2 ~/D/test> dmesg | grep hello  
[ 128.626914] helloworld
```

Kernel space

Step 2: declare

```
1. vim /home/sys_admin/Downloads/linux-5.1 (ssh)  
x vim /home/sys_admin... #1  
unsigned mask, struct statx __user *buffer);  
asm linkage long sys_rseq(struct rseq __user *rseq, uint32_t rseq_len,  
                           int flags, uint32_t sig);  
asm linkage long sys_pidfd_send_signal(int pidfd, int sig,  
                                         siginfo_t __user *info,  
                                         unsigned int flags);  
  
// Project1: new system call  
asm linkage long sys_helloworld(void);
```

Step 1: register

```
1. vim /home/sys_admin/Downloads/linux-5.1 (ssh)  
x vim /home/sys_admin... #1  
328 64 pwritenv  
329 common pkey_mprotect  
330 common pkey_alloc  
331 common pkey_free  
332 common statx  
333 common io_pgetevents  
334 common rseq  
335 common helloworld  
_____  
__x64_sys_pwritenv  
__x64_sys_pkey_mprotect  
__x64_sys_pkey_alloc  
__x64_sys_pkey_free  
__x64_sys_statx  
__x64_sys_io_pgetevents  
__x64_sys_rseq  
__x64_sys_helloworld
```

2. vim /home/sys_admin/Downloads/linux-5.1 (ssh)

```
x vim /home/sys_admin... #1  
do_sysinfo(&val);  
  
if (copy_to_user(info, &val, sizeof(struct sysinfo)))  
    return -EFAULT;  
  
return 0;  
  
// Project1: new system call  
SYSCALL_DEFINE0(helloworld)  
{  
    printk("helloworld");  
    return 0;  
}
```

Step 3: implementation



Outline

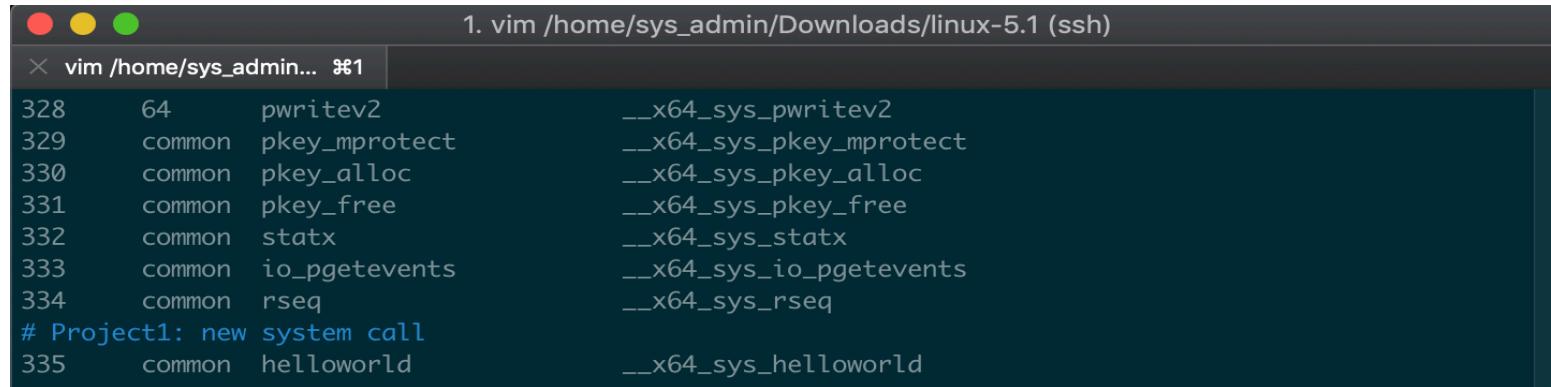
- Assignment 0: create a VM and compile your kernel
- Assignment 1: add a new system call into the Linux kernel
- **Assignment 2: Extend your new system call to print out the calling process's information**
- Assignment 3: Extend your new system call to print out the information of an arbitrary process identified by its PID



Assignment 2:

step 1&2:register and declare

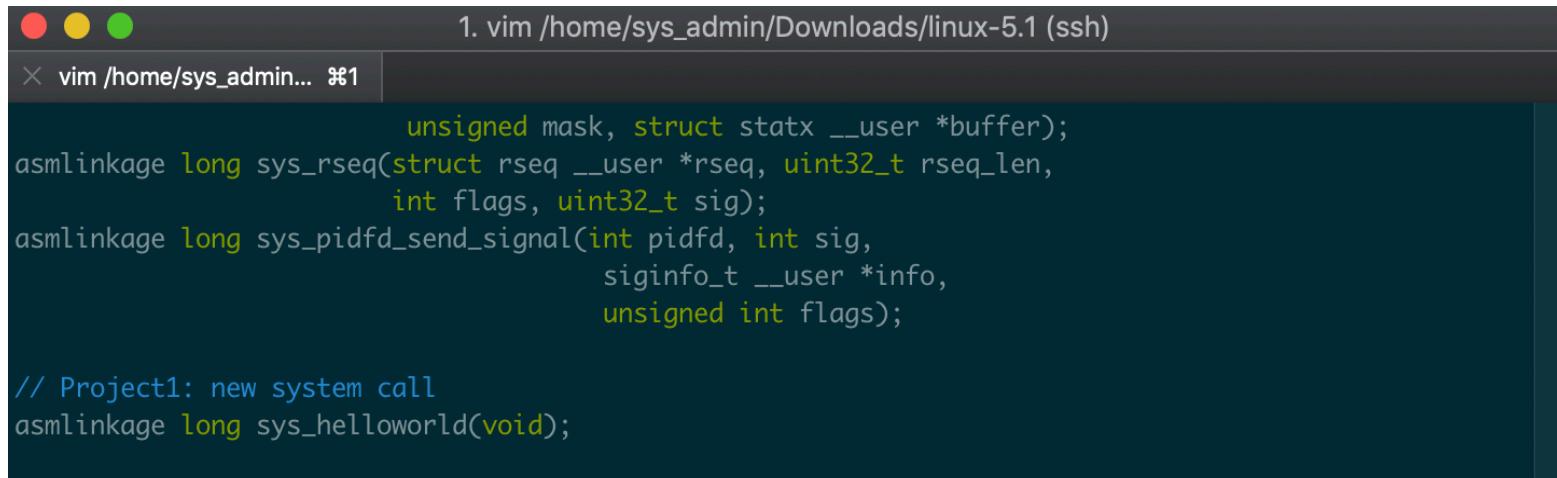
arch/x86/entry/syscalls/syscall_64.tbl



```
1. vim /home/sys_admin/Downloads/linux-5.1 (ssh)
  vim /home/sys_admin... #1

328      64      pwritev2          __x64_sys_pwritev2
329  common  pkey_mprotect      __x64_sys_pkey_mprotect
330  common  pkey_alloc        __x64_sys_pkey_alloc
331  common  pkey_free         __x64_sys_pkey_free
332  common  statx            __x64_sys_statx
333  common  io_pgetevents    __x64_sys_io_pgetevents
334  common  rseq             __x64_sys_rseq
# Project1: new system call
335  common  helloworld       __x64_sys_helloworld
```

include/linux/syscalls.h



```
1. vim /home/sys_admin/Downloads/linux-5.1 (ssh)
  vim /home/sys_admin... #1

              unsigned mask, struct statx __user *buffer);
asmlinkage long sys_rseq(struct rseq __user *rseq, uint32_t rseq_len,
                         int flags, uint32_t sig);
asmlinkage long sys_pidfd_send_signal(int pidfd, int sig,
                                       siginfo_t __user *info,
                                       unsigned int flags);

// Project1: new system call
asmlinkage long sys_helloworld(void);
```

Assignment 2:

step 3: implementation

- print out the following information of the calling process:
 - Process id, running state, and program name
 - Start time and virtual runtime
 - Its parent processes until init (first system process)



Assignment 2:

step 3: implementation

- print out the following information of the calling process:
 - Process id, running state, and program name
 - Start time and virtual runtime
 - Its parent processes until init (first system process)

<https://elixir.bootlin.com/linux/v5.1/source/include/linux/sched.h#L739>



Assignment 2:

step 3: implementation

- print out the following information of the calling process:
 - Process id, running state, and program name
 - Start time and virtual runtime
 - Its parent processes until init (first system process)

<https://elixir.bootlin.com/linux/v5.1/source/include/linux/sched.h#L594>



Assignment 2:

step 3: implementation

- print out the following information of the calling process:
 - Process id, running state, and program name
 - Start time and virtual runtime
 - Its parent processes until init (first system process)

<https://elixir.bootlin.com/linux/v5.1/source/include/linux/sched.h#L840>



Assignment 2:

step 3: implementation

- print out the following information of the calling process:
 - Process id, running state, and program name
 - Start time and virtual runtime
 - Its parent processes until init (first system process)

<https://elixir.bootlin.com/linux/v5.1/source/include/linux/sched.h#L808>



Assignment 2:

step 3: implementation

- print out the following information of the calling process:
 - Process id, running state, and program name
 - Start time and virtual runtime
 - Its parent processes until init (first system process)

<https://elixir.bootlin.com/linux/v5.1/source/include/linux/sched.h#L637>

Task_struct
→ sched_entity
→ vruntime

<https://elixir.bootlin.com/linux/v5.1/source/include/linux/sched.h#L440>



Assignment 2:

step 3: implementation

- print out the following information of the calling process:
 - Process id, running state, and program name
 - Start time and virtual runtime
 - Its parent processes until init (first system process)

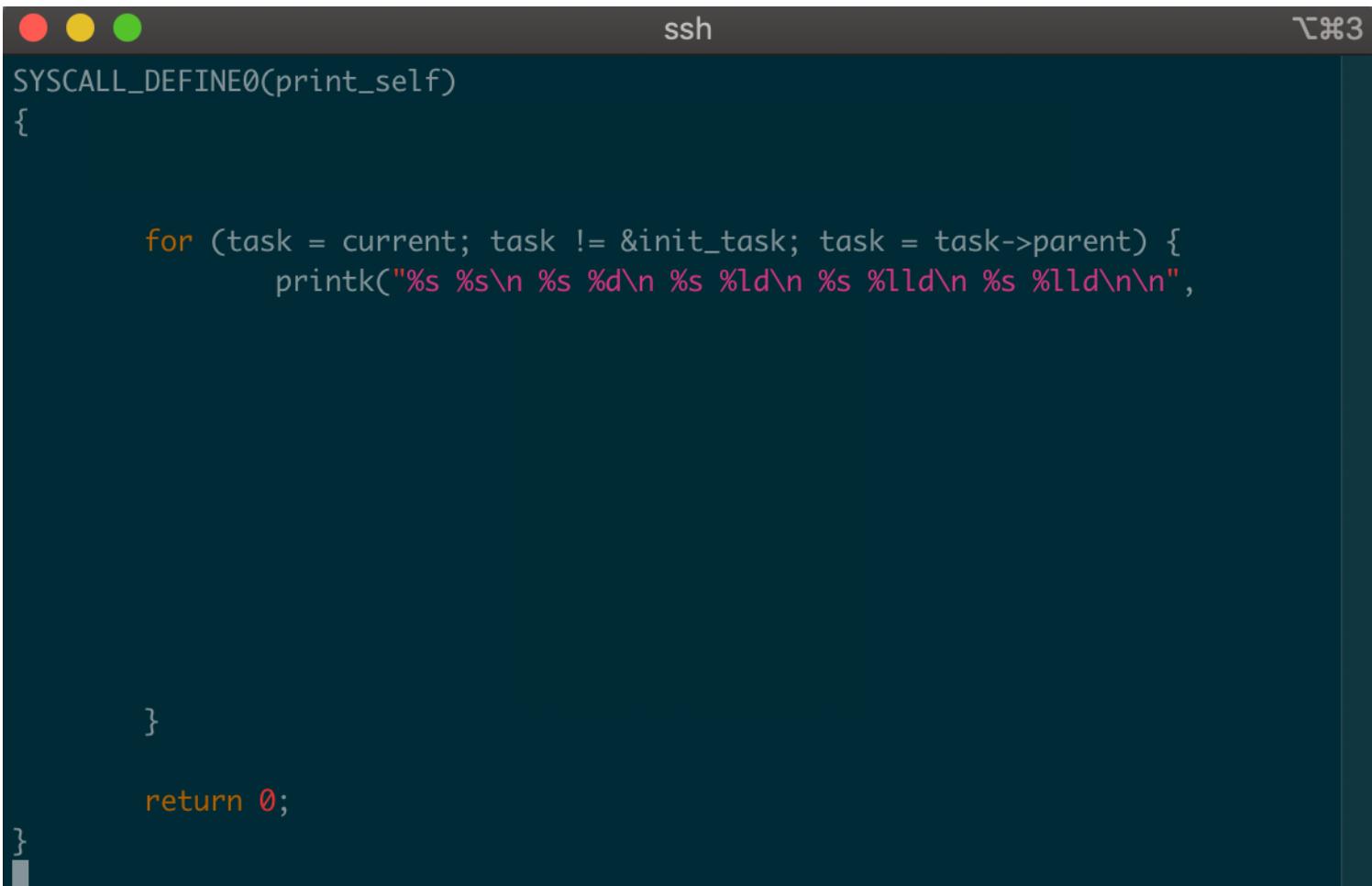
```
pi@raspberrypi ~> pstree -p
systemd(1)─avahi-daemon(304)─avahi-daemon(307)
                     └─{bactl}(683)
                           ├─{gdbus}(685)
                           └─{gmain}(684)
bluealsa(674)─{bactl}(683)
               ├─{gdbus}(685)
               └─{gmain}(684)
bluetoothd(9)
cron(328)
dbus-daemon(305)
dhcpcd(351)
docker(1853)─{docker}(1854)
              ├─{docker}(1855)
              ├─{docker}(1856)
              ├─{docker}(1860)
              └─{docker}(1869)
hciattach(649)
lightdm(458)─Xorg(476)─{InputThread}(488)
               ├─{llvmpipe-0}(482)
               ├─{llvmpipe-1}(483)
               ├─{llvmpipe-2}(484)
               └─{llvmpipe-3}(485)
               └─lxsession(510)─lxpanel(595)─{gdbus}(677)
                           ├─{gdbus}(683)
                           ├─{gmain}(632)
                           └─{menu-cache-io}(762)
               └─lxpolkit(593)─{gdbus}(610)
```

For loop until it reaches the init
{
 printk(...)
}

Assignment 2:

step 3: implementation

kernel/sys.c



The image shows a terminal window titled "ssh" with a dark background. The window contains C code from the file "kernel/sys.c". The code defines a system call "print_self" using the宏 `SYSCALL_DEFINE0`. It prints the task hierarchy starting from the current task until it reaches the initial task, using the `printk` function with a custom format string. The code includes several printf-style specifiers for strings and task IDs.

```
SYSCALL_DEFINE0(print_self)
{
    for (task = current; task != &init_task; task = task->parent) {
        printk("%s %s\n %s %d\n %s %ld\n %s %lld\n %s %lld\n\n",
               task->comm,
               task->name,
               task->pid,
               task->parent->pid,
               task->comm,
               task->id,
               task->parent->id,
               task->comm,
               task->id);
    }
    return 0;
}
```



Assignment 2:

Step 4: write user level app to call it

```
✖ ✎ ✌  
#include <linux/unistd.h>  
#include <sys/syscall.h>  
#include <sys/types.h>  
#include <stdio.h>  
#include <unistd.h>  
  
#define __NR_print_self 336  
  
int main(int argc, char *argv[])  
{  
    syscall (__NR_print_self);  
    return 0;  
}
```

Compile and execute:

```
$ gcc test_syscall2.c -o test_syscall2  
$ ./test_syscall2
```

The test program will call the new system call and output message at the tail of the output of dmesg (system log).

```
$ dmesg
```

At the end of output of \$dmesg

```
sshd:~%2
```

```
[ 66.439077] Take name: test2
  Task PID: 1714
  Task State: 0
  Virtual runtime: 95781496
  Start time: 66428984292

[ 66.439079] Take name: sudo
  Task PID: 1713
  Task State: 1
  Virtual runtime: 49524624
  Start time: 65811247276

[ 66.439081] Take name: fish
  Task PID: 1553
  Task State: 1
  Virtual runtime: 86586173
  Start time: 15521078146

[ 66.439082] Take name: bash
  Task PID: 1542
  Task State: 1
  Virtual runtime: 29803021
  Start time: 14425010066

[ 66.439083] Take name: sshd
  Task PID: 1541
  Task State: 1
  Virtual runtime: 235926572
  Start time: 14421565758

[ 66.439084] Take name: sshd
  Task PID: 1454
  Task State: 1
  Virtual runtime: 280309726
  Start time: 13210824978

[ 66.439086] Take name: sshd
  Task PID: 693
  Task State: 1
  Virtual runtime: 6283579
  Start time: 6391273714

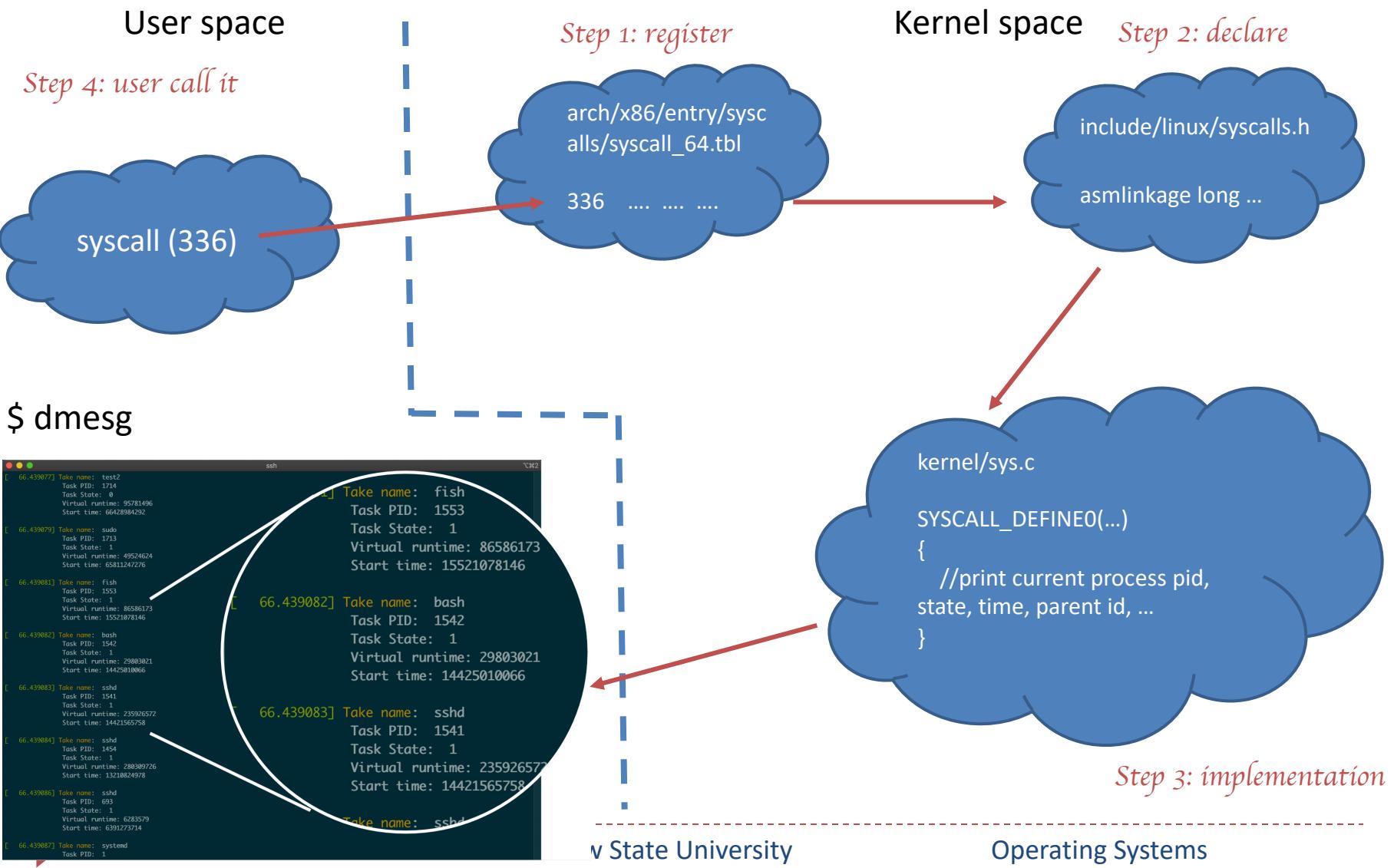
[ 66.439087] Take name: systemd
  Task PID: 1
  Task State: 1
  Virtual runtime: 252410194
  Start time: 211561921
```

The image shows a terminal window titled "ssh" displaying the output of the command \$dmesg. The output lists various kernel tasks with their names, PIDs, states, virtual runtimes, and start times. A large white circle highlights the last four entries in the list:

- Take name: bash
- Task PID: 1542
- Task State: 1
- Virtual runtime: 29803021
- Start time: 14425010066
- Take name: sshd
- Task PID: 1541
- Task State: 1
- Virtual runtime: 235926572
- Start time: 14421565758
- Take name: sshd
- Task PID: 1454
- Task State: 1
- Virtual runtime: 280309726
- Start time: 13210824978
- Take name: systemd
- Task PID: 1
- Task State: 1
- Virtual runtime: 252410194
- Start time: 211561921



Put it all together



Outline

- Assignment 0: create a VM and compile your kernel
- Assignment 1: add a new system call into the Linux kernel
- Assignment 2: Extend your new system call to print out the calling process's information
- **Assignment 3: Extend your new system call to print out the information of an arbitrary process identified by its PID**

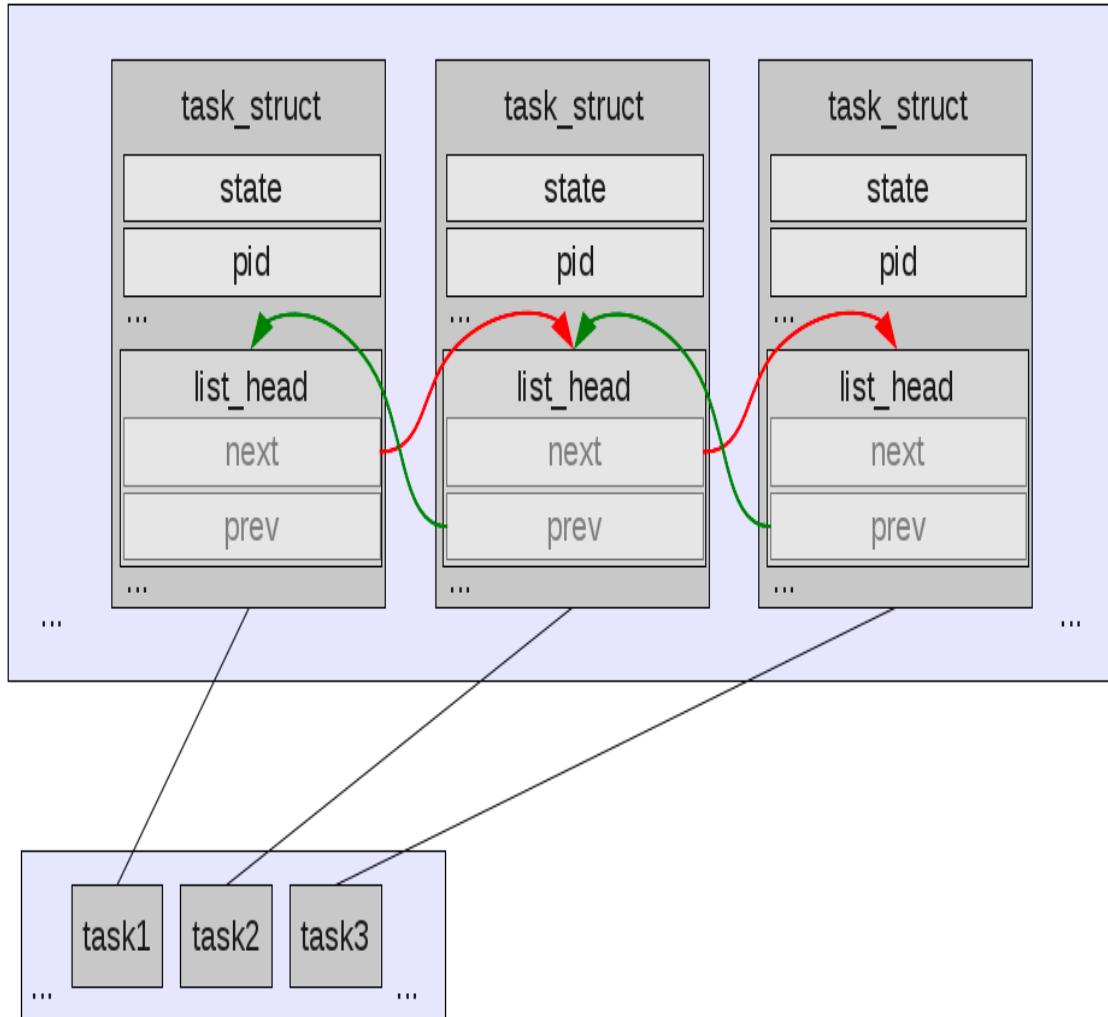


Assignment 3

- `print_other` to print the information for an arbitrary process. The system call takes a process pid as its argument and outputs the above information of this process.
- Input: pid
- Output:
 - Process id, running state, and program name
 - Start time and virtual runtime
 - Its parent processes until init (first system process)



Assignment 3



List_head

<https://elixir.bootlin.com/linux/v5.1/source/include/linux/sched.h#L674>

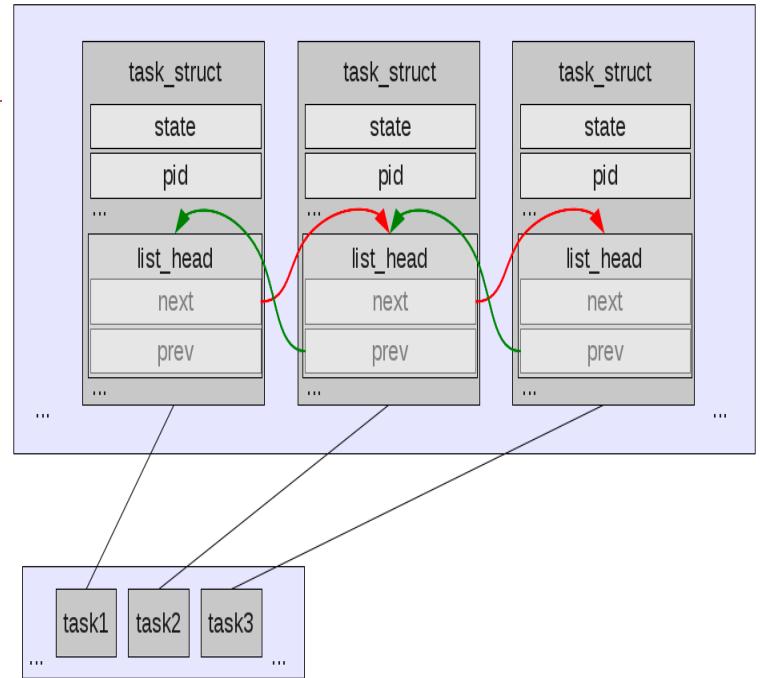
Next, prev

<https://elixir.bootlin.com/linux/v5.1/source/include/linux/types.h#L186>

Assignment 3

```
SYSCALL(int targetpid)
{
    //search pid from the list
    For loop
    {
        if (task->pid == targetpid)
        {
            ...
        }
    }

    //if the pid exists, then print information
    if (exist)
    {
        For loop until it reaches the init
        {
            printk(...)
        }
    }
}
```



Assignment 3

```
SYSCALL(int targetpid)
```

```
{
```

```
    //search pid
```

```
    For loop
```

```
{
```

```
        if (task->pid == targetpid)
```

```
{
```

```
    ...
```

```
Start
```



```
}
```

```
}
```

```
//if the pid exists, then print information
```

```
if (exist)
```

```
{
```

```
    For loop until it reaches the init
```

```
{
```

```
        printk(...)
```

```
}
```

```
}
```

```
CS 3502
```

```
Kennesaw State University
```

Assignment 2

```
Operating Systems
```

Syscall input with parameters

```
SYSCALL_DEFINE0(getpid)
{
    return task_tgid_vnr(current);
}
```

```
SYSCALL_DEFINE1(getpgid, pid_t, pid)
{
    return do_getpgid(pid);
}
```

```
SYSCALL_DEFINE2(sethostname, char __user *, name, int, len)
{
    int errno;
    char tmp[__NEW_UTS_LEN];
```

```
SYSCALL_DEFINE3(setpriority, int, which, int, who, int, niceval)
{
    struct task_struct *g, *p;
    struct user_struct *user;
    const struct cred *cred = current_cred();
    int error = -EINVAL;
```

Step 4: write user level app to call it



```
#include <linux/unistd.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

#define __NR_print_other 337

int main(int argc, char *argv[])
{
    int pid = 30;
    syscall (__NR_print_other, pid);
    return 0;
}
```

Compile and execute:

```
$ gcc test_syscall3.c -o test_syscall3
$ ./test_syscall3
```

The test program will call the new system call and output message at the tail of the output of dmesg (system log).



```
[ 211.659083] Found the process with the pid: 30
[ 211.659086] Task name:
               ksoftirqd/3
               Task PID: 30
               Task State: 1
               Virtual runtime: 20450083624
               Start time: 215561921
```

```
[ 211.659087] Task name:
               kthreadd
               Task PID: 2
               Task State: 1
               Virtual runtime: 4781497360
               Start time: 211561921
```



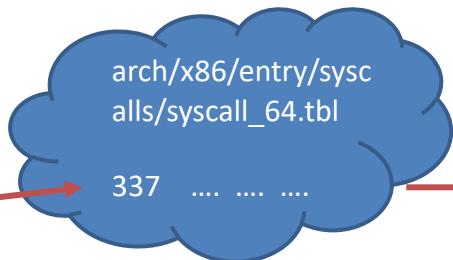
Put it all together

User space

Step 4: user call it

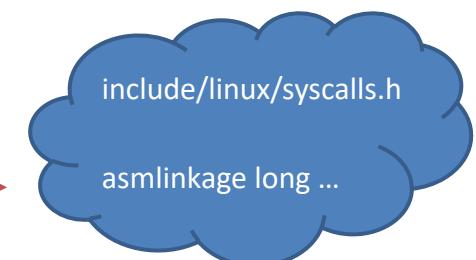


Step 1: register



Kernel space

Step 2: declare



\$ dmesg

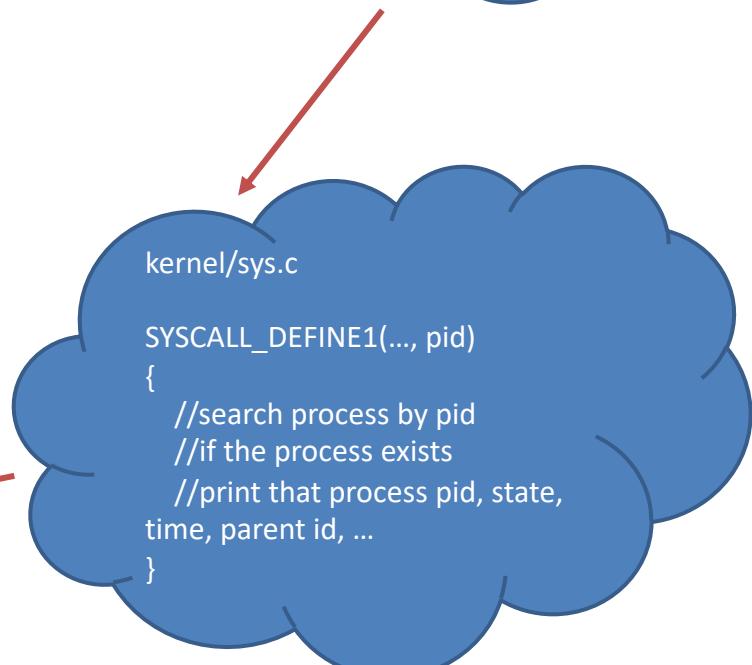
```
[ 211.659083] Found the process with the pid: 30
[ 211.659086] Task name:
              ksoftirqd/3
              Task PID: 30
              Task State: 1
              Virtual runtime: 20450083624
              Start time: 215561921

[ 211.659087] Task name:
              kthreadd
              Task PID: 2
              Task State: 1
              Virtual runtime: 4781497360
              Start time: 211561921
```

State University

Operating Systems

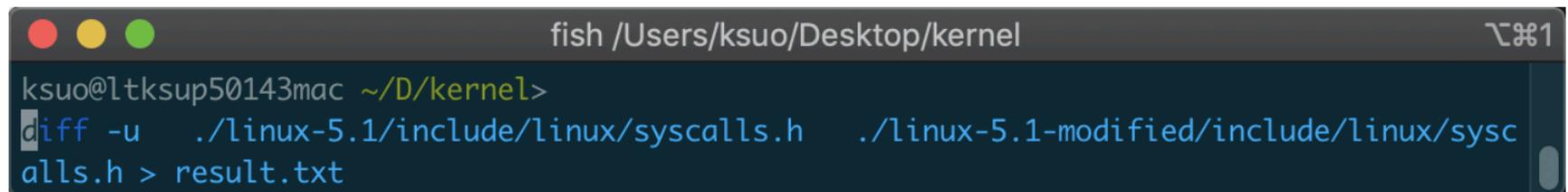
Step 3: implementation



Submission

Please use diff command to highlight your modification:
\$ diff -u original_file.c modified_file.c > result.txt

For example, to show the difference between file
include/linux/syscalls.h, just use the command below:



A screenshot of a macOS terminal window titled "fish /Users/ksuo/Desktop/kernel". The window shows the command "diff -u ./linux-5.1/include/linux/syscalls.h ./linux-5.1-modified/include/linux/syscalls.h > result.txt" being run. The terminal is dark-themed with red, yellow, and green window controls.

```
fish /Users/ksuo/Desktop/kernel
ksuo@ltksup50143mac ~/D/kernel>
diff -u ./linux-5.1/include/linux/syscalls.h ./linux-5.1-modified/include/linux/syscalls.h > result.txt
```



result.txt content

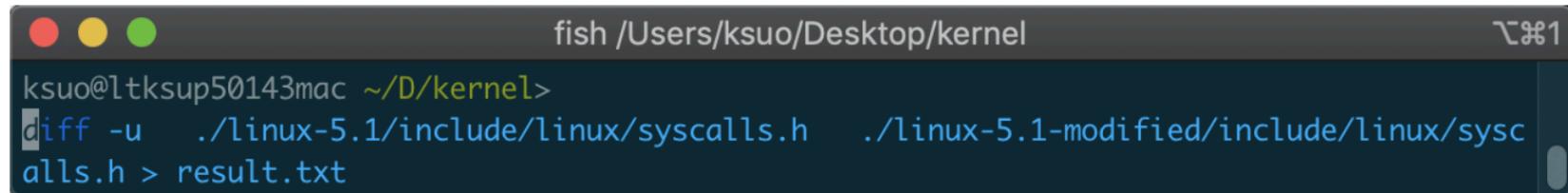
```
vm1 [Running]
Activities Terminal Tue 18:41
vim /home/ksuo/Downloads
File Edit View Search Terminal Help
--- ./linux-5.1.1/arch/x86/entry/syscalls/syscall_64.tbl      2019-05-11 01:4
9:56.000000000 -0400
+++ ./linux-5.1.1-modified/arch/x86/entry/syscalls/syscall_64.tbl    2019-09
-08 10:16:11.011187097 -0400
@@ -343,6 +343,10 @@
 332 common statx          __x64_sys_statx
 333 common io_pgetevents  __x64_sys_io_pgetevents
 334 common rseq            __x64_sys_rseq
+335 common helloworld     __x64_sys_helloworld
+
+
+
# don't use numbers 387 through 423, add new calls after the last
# 'common' entry
424 common pidfd_send_signal __x64_sys_pidfd_send_signal
~
~
```



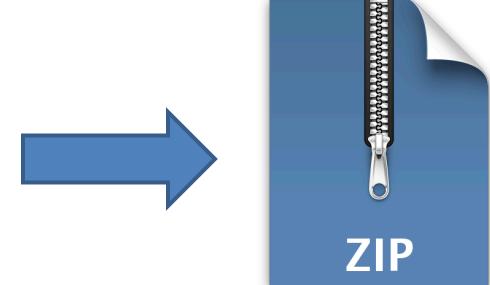
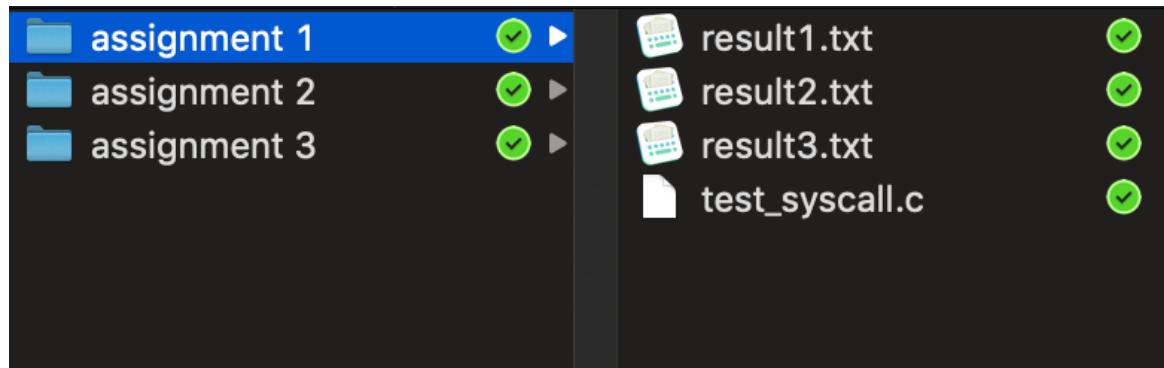
Submission

Please use diff command to highlight your modification:

```
$ diff -u original_file.c modified_file.c > result.txt
```



```
fish /Users/ksuo/Desktop/kernel
ksuo@ltksup50143mac ~/D/kernel>
diff -u ./linux-5.1/include/linux/syscalls.h ./linux-5.1-modified/include/linux/syscalls.h > result.txt
```



CS3502_D2Lname.zip



Questions

- T/Th 3-4pm, J-318
- Send me emails or make appointments



Print in C

- User space
 - `printf("%d", variable);`
 - `printf("%d", 1234);`
- Kernel space
 - `printk("%d", 1234);`



Print in C

%c	character
%d	decimal (integer) number (base 10)
%f	floating-point number
%s	a string of characters
%u	unsigned decimal (integer) number
...	...

```
printf("%d", 1234);
printf("%c", a);
printf("%f", -1.234);
printf("%s", "helloworld");
...
...
```



Print in C

If variable is of Type, use printk format specifier:

int	%d or %x
unsigned int	%u or %x
long	%ld or %lx
unsigned long	%lu or %lx
long long	%lld or %llx
unsigned long long	%llu or %llx
size_t	%zu or %zx
ssize_t	%zd or %zx
s32	%d or %x
u32	%u or %x
s64	%lld or %llx
u64	%llu or %llx

<https://elixir.bootlin.com/linux/v5.1/source/include/linux/sched.h#L808>

```
printf("%llu", start_time);
```



Print in C

If variable is of Type, use printk format specifier:

int	%d or %x
unsigned int	%u or %x
long	%ld or %lx
unsigned long	%lu or %lx
long long	%lld or %llx
unsigned long long	%llu or %llx
size_t	%zu or %zx
ssize_t	%zd or %zx
s32	%d or %x
u32	%u or %x
s64	%lld or %llx
u64	%llu or %llx

<https://elixir.bootlin.com/linux/v5.1/source/include/linux/sched.h#L594>

```
printf("%ld", state);
```

