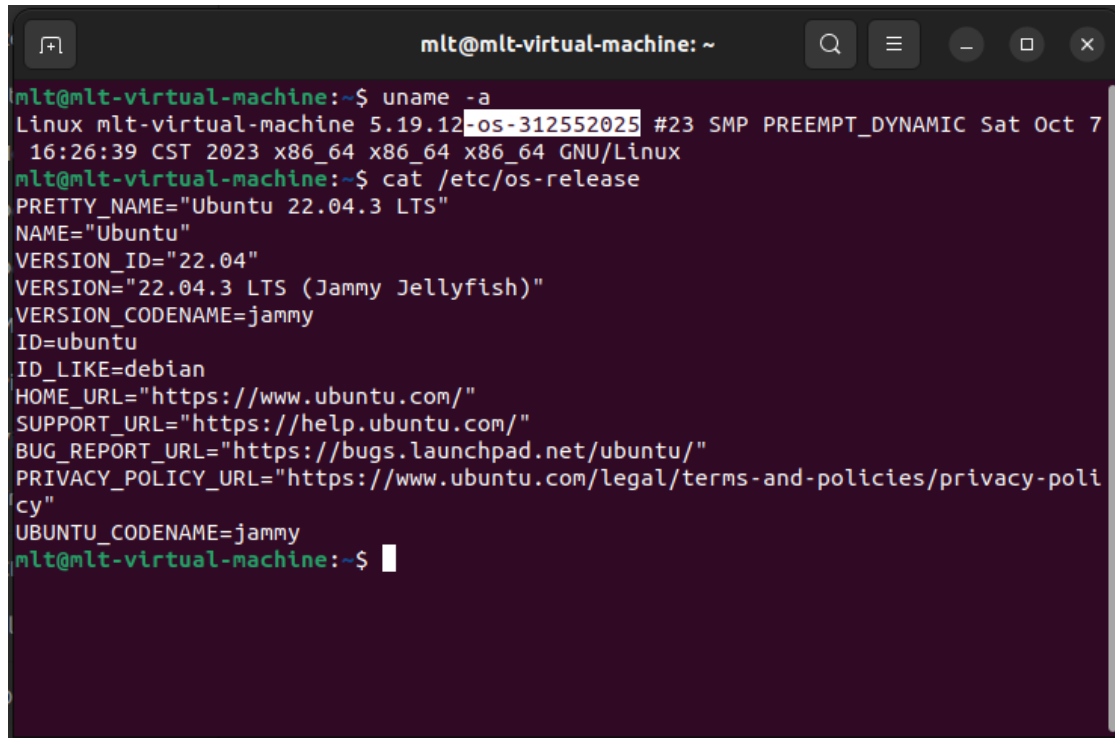


# 1121 OS-Assignment 1

312552025 網工所 蔡明霖

## I. Compiling Linux Kernel

A terminal window titled 'mlt@mlt-virtual-machine: ~' with standard window controls. The terminal shows the output of 'uname -a' and 'cat /etc/os-release'. The 'uname -a' output includes the IP address '10.0.2.15' which has been redacted with a white box. The 'cat /etc/os-release' output shows Ubuntu 22.04.3 LTS (Jammy Jellyfish) information.

```
mlt@mlt-virtual-machine:~$ uname -a
Linux mlt-virtual-machine 5.19.12-os-312552025 #23 SMP PREEMPT_DYNAMIC Sat Oct 7
16:26:39 CST 2023 x86_64 x86_64 x86_64 GNU/Linux
mlt@mlt-virtual-machine:~$ cat /etc/os-release
PRETTY_NAME="Ubuntu 22.04.3 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="22.04.3 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=jammy
mlt@mlt-virtual-machine:~$
```

## II. Adding Custom System Calls

### 1. Build the system call program:

First, we have to build two C program for our custom System calls.

Kernel programs have few constraints in contrast with user-space programs, including:

- (1) We can't use C library directly like what we usually do in user-space programs: Use GNU core C libraries(i.e. GNU C or glibc) instead.
- (2) We need to handle the data transfer between user-space and kernel-space: Use 'copy\_from\_user()' to handle memory access between user space and kernel.
- (3) We need to find a way that allows the kernel to recognize our system call: Modify files in the following steps.

Following codes are the implementation of my system calls:

#### (1) syscall\_hello.c

```

#include <linux/kernel.h>
#include <linux/syscalls.h>

/*Entry point for my system call*/
SYSCALL_DEFINE0(hello){
    printk("Hello, world!\n");
    printk("312552025\n");

    return 0;
}

```

## (2) syscall\_revstr.c

```

/* It's using C90 by default, and in C90 there's no "/*"
comment style in it*/
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/slab.h> /* for kmalloc */
#include <linux/string.h>

/* SYSCALL_DEFINE1(user_space_func_name, 1st
parameter type, 1st parameter name)*/
/* char __user *str: get string from user space */
SYSCALL_DEFINE2(revstr, int, len, const char __user *,
str){
    char *rev_str = kmalloc(len + 1, GFP_KERNEL);
    char *dup_str = kmalloc(len + 1, GFP_KERNEL);
    int i = 0; /* index for rev_str */

    /* Warning would appear if this block put in the
begining of function, need to dig in how to use other
like C99 to compile */
    if(!str)
        return -1;

    if(!rev_str || !dup_str){
        printk("kmalloc error\n");
        return -1;
    }
}

```

```

if(copy_from_user(dup_str, str, len + 1) )
    return -1;

printk("The origin string: %s\n", dup_str);

/* there seems no "strrev()" function in
<linux/string.h> library*/
len--; /*the "\0" always at the end*/
while(len >= 0){
    rev_str[l] = dup_str[len];
    l++;
    len--;
}
rev_str[l] = '\0';

printk("The reversed string: %s\n", rev_str);

return 0;
}

```

## 2. Modify kernel\_path/include/linux/syscall.h:

This step is to add the function prototypes(declarations) of our custom system calls, so whenever a file include this header file and invoke our system call, our systems call functions can be found.

```
hw1_1.c  syscall_hello.c 2  syscall_revstr.c 3  sys_ni.c 2  unistd_64.h  C
home > mlt > Desktop > os > linux-5.19.12 > include > linux > C syscalls.h > ...
1265
1266 /* obsolete: mm/ */
1267 asmlinkage long sys_mmap_pgoff(unsigned long addr, unsigned long len,
1268                                unsigned long prot, unsigned long flags,
1269                                unsigned long fd, unsigned long pgoff);
1270 asmlinkage long sys_old_mmap(struct mmap_arg_struct __user *arg);
1271
1272
1273 /*
1274  * Not a real system call, but a placeholder for syscalls which are
1275  * not implemented -- see kernel/sys_ni.c
1276  */
1277 asmlinkage long sys_ni_syscall(void);
1278
1279 /*****
1280 Function prototype for the system-call of 1121 OS assignment1
1281 *****/
1282 asmlinkage long sys_hello(void);
1283 asmlinkage long sys_revstr(int len, char __user *str);
1284
1285 #endif /* CONFIG_ARCH_HAS_SYSCALL_WRAPPER */
1286
1287
1288 /*
1289  * Kernel code should not call syscalls (i.e. sys_xyzvz()) directly
1290  */
```

### 3. Modify kernel\_path/arch/x86/entry/syscalls/syscall\_64.tbl:

We add two new system call table entries('sys\_hello' and 'sys\_revstr') into the generic table, and give our system calls with unused numbers (550 and 551, since the 32bits system call numbers end at 547 in this kernel version, we have to choose number larger than that).

This file would be used to bind syscall number and our system calls, and the numbers in this file would be looked up(or through other files generated based on this file, like 'syscalls\_64.h') when a system call interrupt is invoked, allowing kernel to find out which system call service routine should be executed.

```
home > mlt > Desktop > os > linux-5.19.12 > arch > x86 > entry > syscalls > syscall_64.tbl
405 535 x32 pwritev      compat_sys_pwritev64
406 536 x32 rt_tsigqueueinfo compat_sys_rt_tsigqueueinfo
407 537 x32 recvmmsg     compat_sys_recvmmsg_time64
408 538 x32 sendmmsg     compat_sys_sendmmsg
409 539 x32 process_vm_readv sys_process_vm_readv
410 540 x32 process_vm_writev sys_process_vm_writev
411 541 x32 setsockopt    sys_setsockopt
412 542 x32 getsockopt    sys_getsockopt
413 543 x32 io_setup      compat_sys_io_setup
414 544 x32 io_submit     compat_sys_io_submit
415 545 x32 execveat      compat_sys_execveat
416 546 x32 preadv2       compat_sys_preadv64v2
417 547 x32 pwritev2      compat_sys_pwritev64v2
418 # This is the end of the legacy x32 range. Numbers 548 and above are
419 # not special and are not to be used for x32-specific syscalls.
420
421 #update the master syscall tables for my system call.
422 550 common hello      sys_hello
423 551 common revstr     sys_revstr
```

#### 4. Modify kernel\_path/kernel/sys\_ni.c:

This file is to handle those system calls no longer used(i.e. not-implemented), their syscall number won't be deleted, but instead they would be bind to the system calls defined in sys\_ni.c. Which usually just return an error message. Here we put our system calls in it to prevent from the kernel unable to find our implementations.

```

home > mlt > Desktop > os > linux-5.19.12 > kernel > C sys_ni.c > ...
467 COND_SYSCALL(getresuid16);
468 COND_SYSCALL(getuid16);
469 COND_SYSCALL(lchown16);
470 COND_SYSCALL(setfsgid16);
471 COND_SYSCALL(setfsuid16);
472 COND_SYSCALL(setgid16);
473 COND_SYSCALL(setgroups16);
474 COND_SYSCALL(setregid16);
475 COND_SYSCALL(setresgid16);
476 COND_SYSCALL(setresuid16);
477 COND_SYSCALL(setreuid16);
478 COND_SYSCALL(setuid16);
479
480 /* restartable sequence */
481 COND_SYSCALL(rseq);
482
483 /*Fallback stub implementation of my system call*/
484 COND_SYSCALL(hello);
485 COND_SYSCALL(revstr);

```

## 5. Build a new Makefile in kernel\_path/hw1

This makefile is put under the same folder as our system call implementations. When the configs of our system call (refer to step 7) is activated, this makefile would compile our system calls into the new kernel.

```

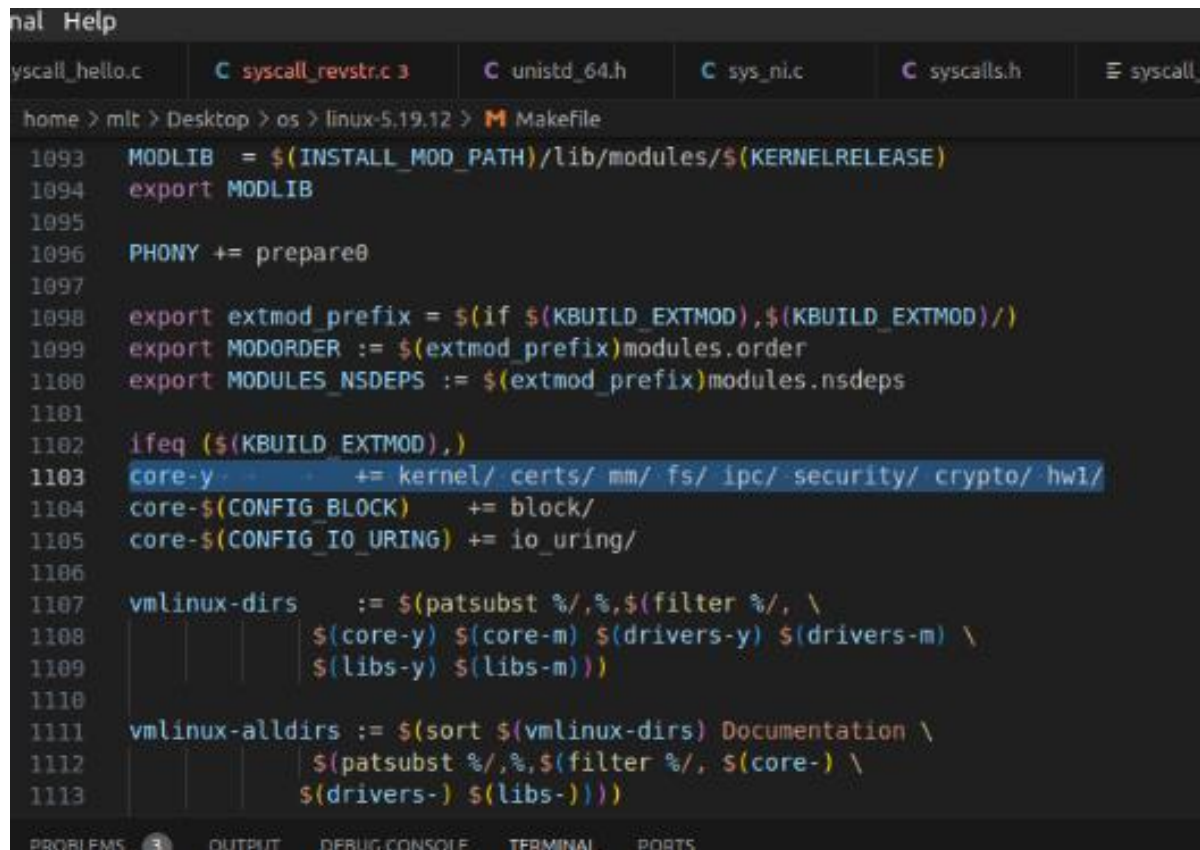
+ 10 21:20
Makefile - hw1 - Visual Studio Code
ninal Help
call_revstr.c 3 C unistd_64.h C sys_ni.c C syscalls.h syscall_64.tbl Kconfig
home > mlt > Desktop > os > linux-5.19.12 > hw1 > M Makefile
1 #Add options to allow user to decide whether to compile the customs system calls
2 obj-$(CONFIG_HELLO) += syscall_hello.o
3 obj-$(CONFIG_REVSTR) += syscall_revstr.o
4
5

```

## 6. Modify kernel\_path/Makefile:

Since I put my system call programs in the kernel\_path/hw1

folder(I also tried to put inside the “init” folder, and it worked even without modifying this makefile), so I must add path to the ‘core-y’ in Makefile to let my system call programs can be correctly detected and compiled.



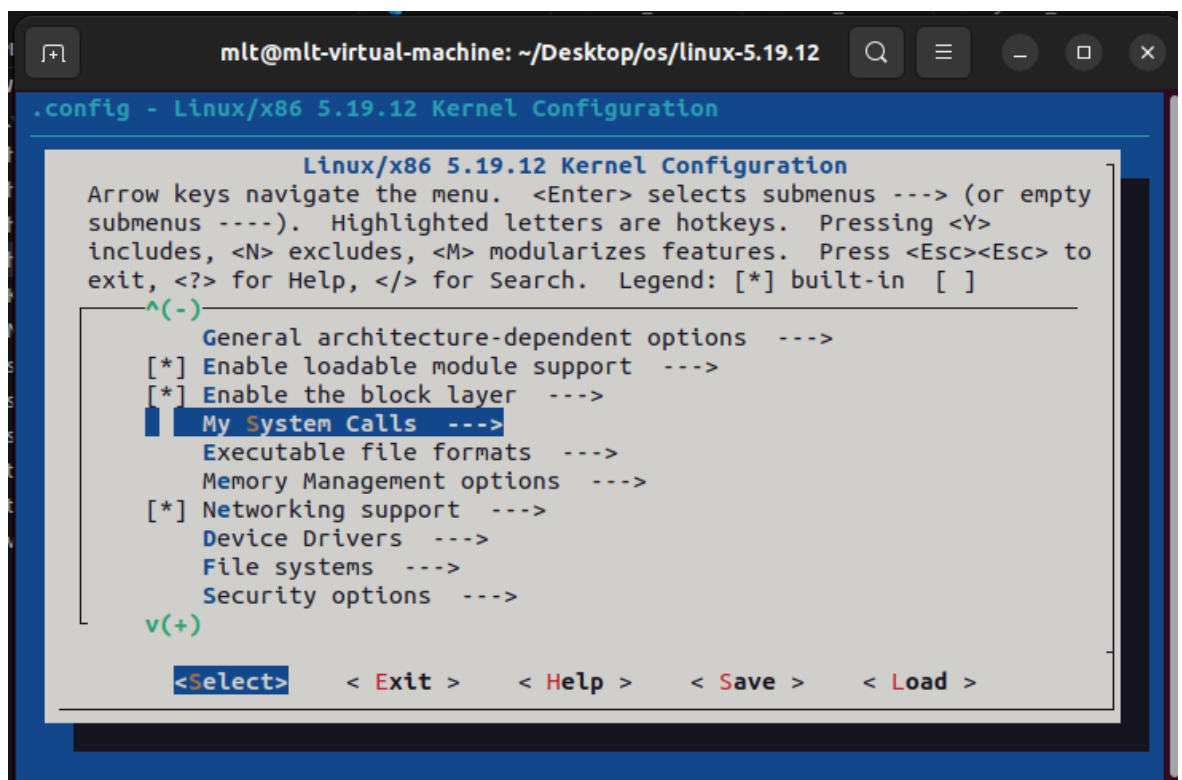
```
1093 MODLIB = $(INSTALL_MOD_PATH)/lib/modules/$(KERNELRELEASE)
1094 export MODLIB
1095
1096 PHONY += prepare0
1097
1098 export extmod_prefix = $(if $(KBUILD_EXTMOD),$(KBUILD_EXTMOD)/)
1099 export MODORDER := $(extmod_prefix)modules.order
1100 export MODULES_NSDEPS := $(extmod_prefix)modules.nsdeps
1101
1102 ifeq ($(KBUILD_EXTMOD),)
1103 core-y += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ hw1/
1104 core-$(CONFIG_BLOCK) += block/
1105 core-$(CONFIG_IO_URING) += io_uring/
1106
1107 vmlinux-dirs := $(patsubst %/,%, $(filter %/, \
1108 $(core-y) $(core-m) $(drivers-y) $(drivers-m) \
1109 $(libs-y) $(libs-m)))
1110
1111 vmlinux-alldirs := $(sort $(vmlinux-dirs) Documentation \
1112 $(patsubst %/,%, $(filter %/, $(core-) \
1113 $(drivers-) $(libs-))))
```

## 7. Modify kernel\_path/init/Kconfig for menuconfig:

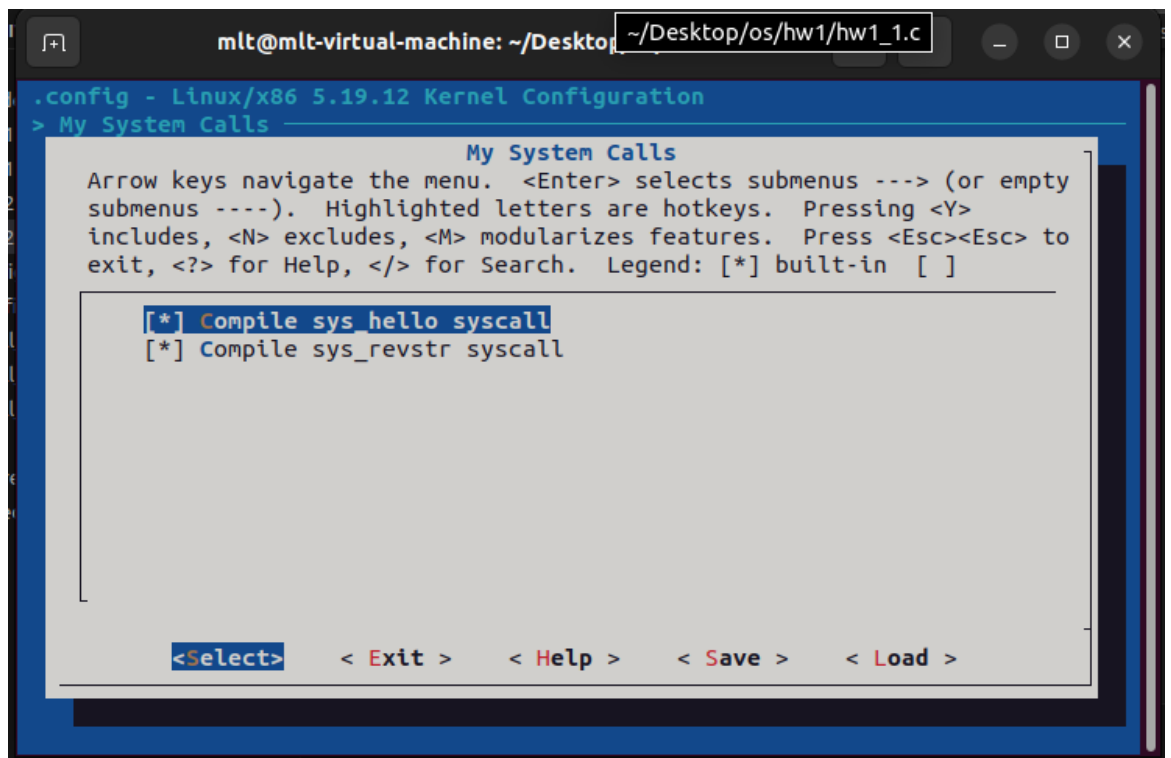
As the tutorial mentioned, we should make our custom system calls be optional. To achieve that, the best choice would probably be “menuconfig”, which is a GUI used by many people to compile their own kernel.

The menuconfig would read file “Kconfig” to generate the contents of GUI, so I modify it to add a new menu. Inside the menu there’re two boolean configurations, allowing users to decide whether to compile my system calls.

```
w1_1.c  C hw1_2.c  C syscall_hello.c 2  C syscall_revstr.c 3  C test_revstr.c  C sys_ni.c 2
home > mlt > Desktop > os > linux-5.19.12 > init > M Kconfig
2250     bool
2251
2252     #CONFIG for custom system calls
2253
2254     menu "My System Calls"
2255
2256     config HELLO
2257     ... bool "Compile sys_hello syscall"
2258     ... default n
2259     ... help
2260     ... Choose whether to compile sys_hello syscall
2261
2262     config REVSTR
2263     ... bool "Compile sys_revstr syscall"
2264     ... default n
2265     ... help
2266     ... Choose whether to compile sys_revstr syscall
2267
2268     endmenu
2269
2270     # It may be useful for an architecture to override the definitions of the
2271     # SYSCALL_DEFINE() and SYSCALL_DEFINE() macros in <linux/syscalls.h>
```







## 8. Recompile kernel:

- (1) First go to the kernel folder(`cd ~/Desktop/OS/linux-5.19.12` in my case)
- (2) type '`make menuconfig`' to activate the options of my system calls as the pictures of last step showed.
- (3) Then type '`sudo make -j8 > /dev/null && sudo make modules_install install && sudo update-grub2`' to :
  - i. compile the kernel,
  - ii. make modules and install modules
  - iii. install the new kernel
  - iv. update the grub(I modify the grub config elapsed time to 15 seconds allowing me to choose kernel more convenient).

## 9. Run test programs provided by TA:

First check the syscall number in `kernel_path/arch/x86/include/generated/uapi/asm/unistd_64.h` to make sure that our system call can be correctly invoked using those syscall numbers.

```

Welcome  C hw1_1.c  C hw1_2.c  C syscall_hello.c  C syscall_revstr.c  C sy
home > mlt > Desktop > os > linux-5.19.12 > arch > x86 > include > generated > uapi > asm > C unistd_64.h > E
354 #define __NR_faccessat2 439
355 #define __NR_process_madvise 440
356 #define __NR_epoll_pwait2 441
357 #define __NR_mount_setattr 442
358 #define __NR_quotactl_fd 443
359 #define __NR_landlock_create_ruleset 444
360 #define __NR_landlock_add_rule 445
361 #define __NR_landlock_restrict_self 446
362 #define __NR_memfd_secret 447
363 #define __NR_process_mrelease 448
364 #define __NR_futex_waitv 449
365 #define __NR_set_mempolicy_home_node 450
366 #define __NR_hello 550
367 #define __NR_revstr 551
368
369 #ifdef __KERNEL__
370 #define __NR_syscalls 552
371 #endif
372
373 #endif /* _UAPI_ASM_UNISTD_64_H */
374
```

Then compile the C codes provided by TA using gcc, then execute them. To check the outputs of the programs, we must use '*sudo dmesg / tail*' to show the latest log messages.

(1) sys\_hello:

+ 14 21:58

hw1\_1.c - hw1 - Visual Studio Code

inal Help

Welcome

C hw1\_1.c

C hw1\_2.c

C syscall\_hello.c 2

C syscall\_revstr.c

C sys

C hw1\_1.c

> main(int, char \* [])

1 #include <assert.h>

2 #include <unistd.h>

3 #include <sys/syscall.h>

4

5 /\*

6 \* You must copy the \_\_NR\_hello marco from

7 \* <your-kernel-build-dir>/arch/x86/include/generated/uapi/asam/unistd\_64.h

8 \* In this example, the value of \_\_NR\_hello is 548

9 \*/

10 #define \_\_NR\_hello 550

11

12 int main(int argc, char \*argv[]) {

13 int ret = syscall(\_\_NR\_hello);

14 assert(ret == 0);

15

16 return 0;

17 }

PROBLEMS 2

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

● mlt@mlt-virtual-machine:~/Desktop/os/hw1\$ ./hw1\_1

● mlt@mlt-virtual-machine:~/Desktop/os/hw1\$ sudo dmesg | tail

[ 185.517015] Hello, world!

[ 185.517018] 312552025

[ 187.576048] The origin string: hello

[ 187.576051] The reversed string: olleh

[ 187.576052] The origin string: 5Y573M C411

[ 187.576053] The reversed string: 114C M375Y5

[ 244.558988] Hello, world!

[ 244.558990] 312552025

[ 355.558784] Hello, world!

[ 355.558786] 312552025

● mlt@mlt-virtual-machine:~/Desktop/os/hw1\$ ./hw1\_1

● mlt@mlt-virtual-machine:~/Desktop/os/hw1\$ sudo dmesg | tail -2

[ 374.356495] Hello, world!

[ 374.356498] 312552025

○ mlt@mlt-virtual-machine:~/Desktop/os/hw1\$

(2) sys\_revstr:

inal Help

Welcome

C hw1\_1.c

C hw1\_2.c

C syscall\_hello.c 2

C syscall\_revstr.c

C sys\_

C hw1\_2.c &gt; \_\_NR\_revstr

```
1 #include <assert.h>
2 #include <unistd.h>
3 #include <sys/syscall.h>
4
5 /*
6  * You must copy the __NR_revstr marco from
7  * <your-kernel-build-dir>/arch/x86/include/generated/uapi/asam/unistd_64.h
8  * In this example, the value of __NR_revstr is 549
9  */
10 #define __NR_revstr 551
11
12 int main(int argc, char *argv[]) {
13     int ret1 = syscall(__NR_revstr, 5, "hello");
14     assert(ret1 == 0);
15
16     int ret2 = syscall(__NR_revstr, 11, "5Y573M C411");
17     assert(ret2 == 0);
18
19     return 0;
20 }
```

PROBLEMS 2

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

```
[ 187.576052] The origin string: 5Y573M C411
[ 187.576053] The reversed string: 114C M375Y5
[ 244.558988] Hello, world!
[ 244.558990] 312552025
[ 355.558784] Hello, world!
[ 355.558786] 312552025
● mlt@mlt-virtual-machine:~/Desktop/os/hw1$ ./hw1_1
● mlt@mlt-virtual-machine:~/Desktop/os/hw1$ sudo dmesg | tail -2
[ 374.356495] Hello, world!
[ 374.356498] 312552025
● mlt@mlt-virtual-machine:~/Desktop/os/hw1$ ./hw1_2
● mlt@mlt-virtual-machine:~/Desktop/os/hw1$ sudo dmesg | tail -4
[ 394.140524] The origin string: hello
[ 394.140528] The reversed string: olleh
[ 394.140529] The origin string: 5Y573M C411
[ 394.140530] The reversed string: 114C M375Y5
● mlt@mlt-virtual-machine:~/Desktop/os/hw1$
```