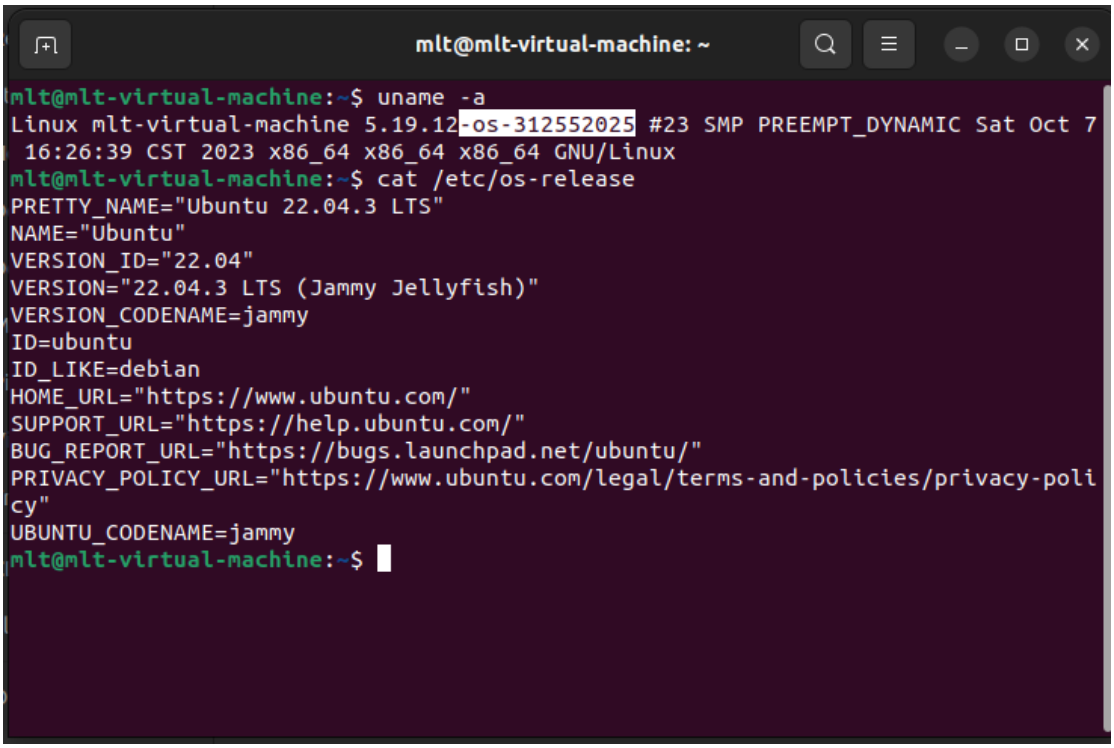
1121 OS-Assignment 1

312552025 網工所 蔡明霖

I. Compiling Linux Kernel



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

```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;
}
```

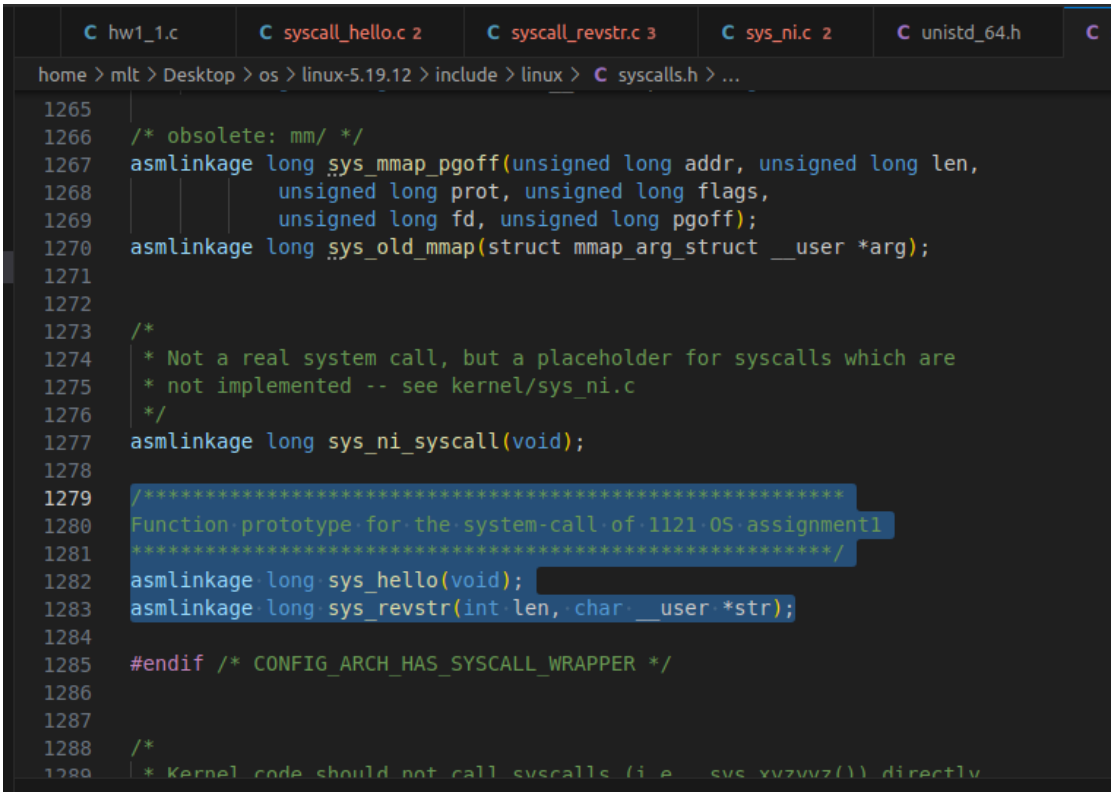
(2) syscall_revstr.c

```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 l = 0; /* index for rec_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:

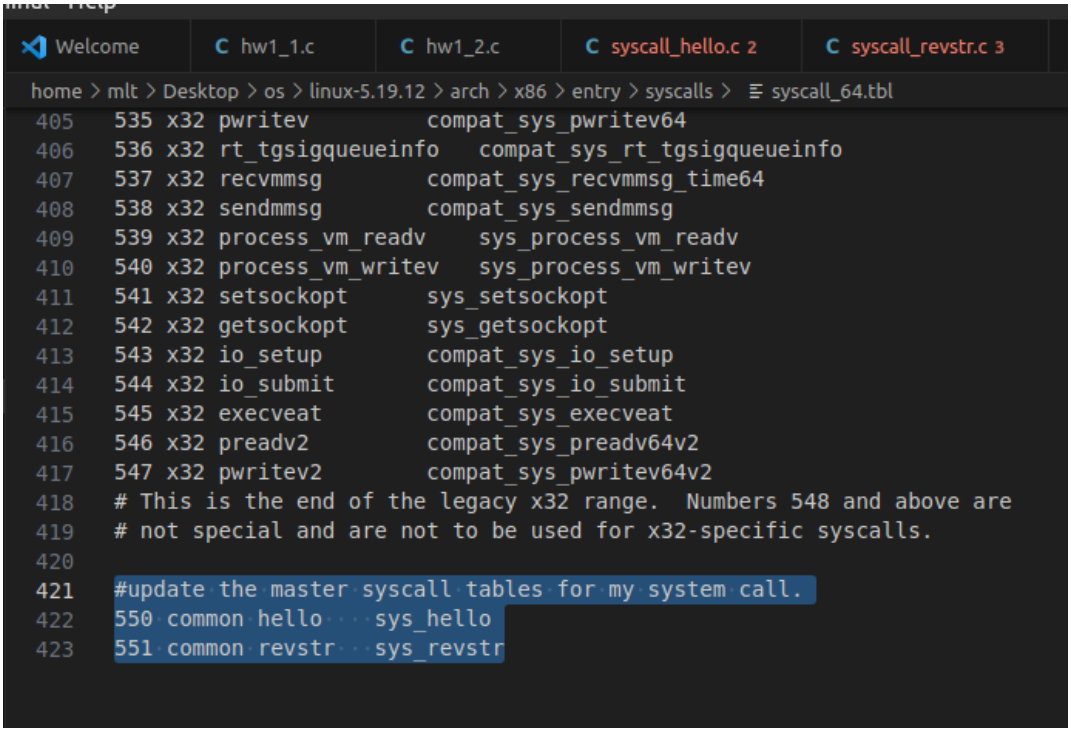
This step is to add the function prototypes(declarations) of our custom system calls, so whenever a file include this header file and invoke our system call, our systems call functions can be found.



3. Modify kernel_path/arch/x86/entry/syscalls/syscall_64.tbl:

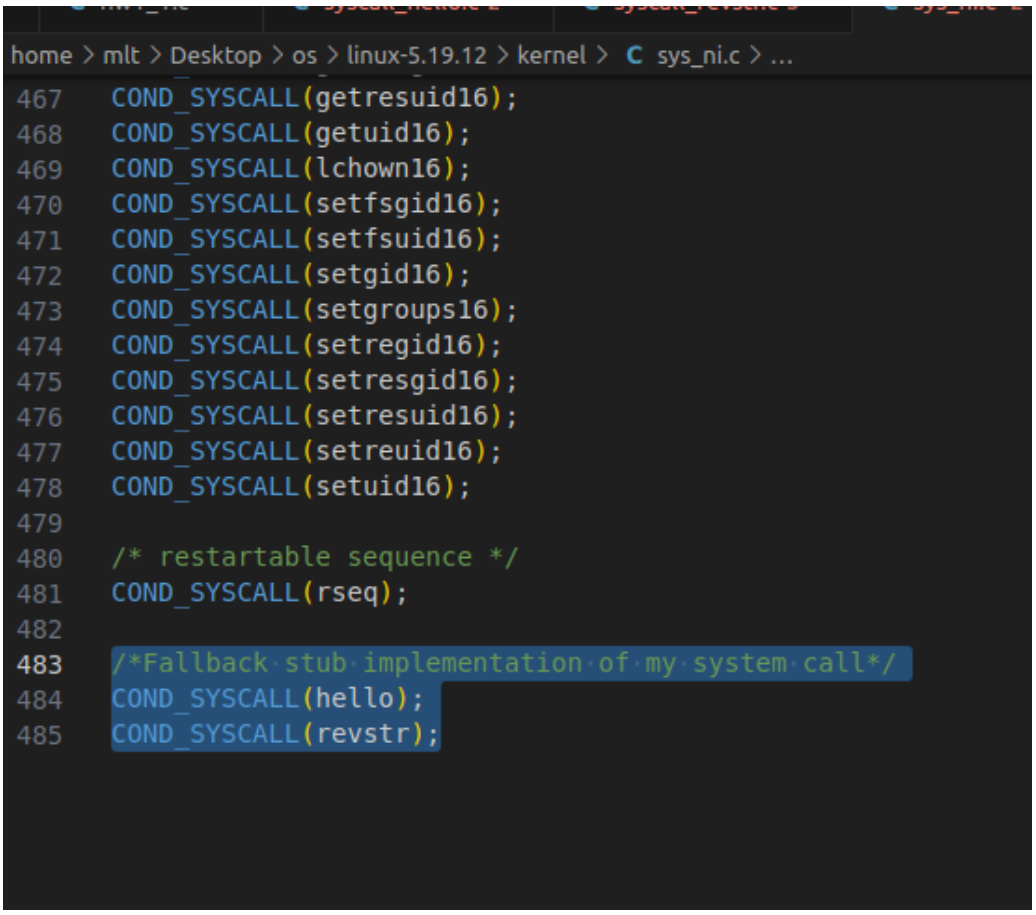
We add two new system call table entries(‘sys_hello’ and ‘sys_revstr’) into the generic table, and give our system calls with unused numbers (550 and 551, since the 32bits system call numbers end at 547 in this kernel version, we have to choose number larger than that).

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.



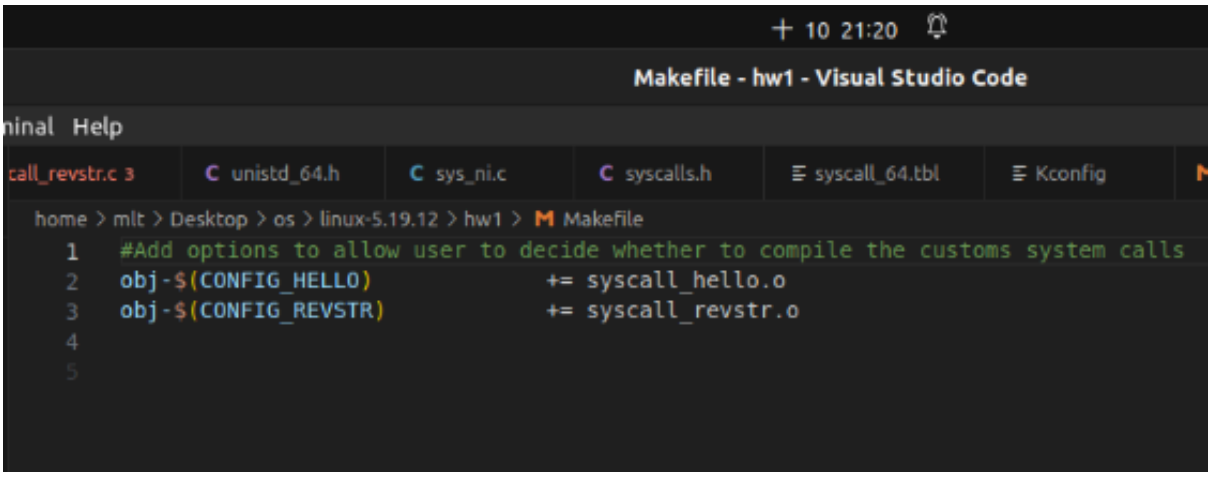
4. Modify kernel_path/kernel/sys_ni.c:

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.



5. **Build a new Makefile in kernel_path/hw1**

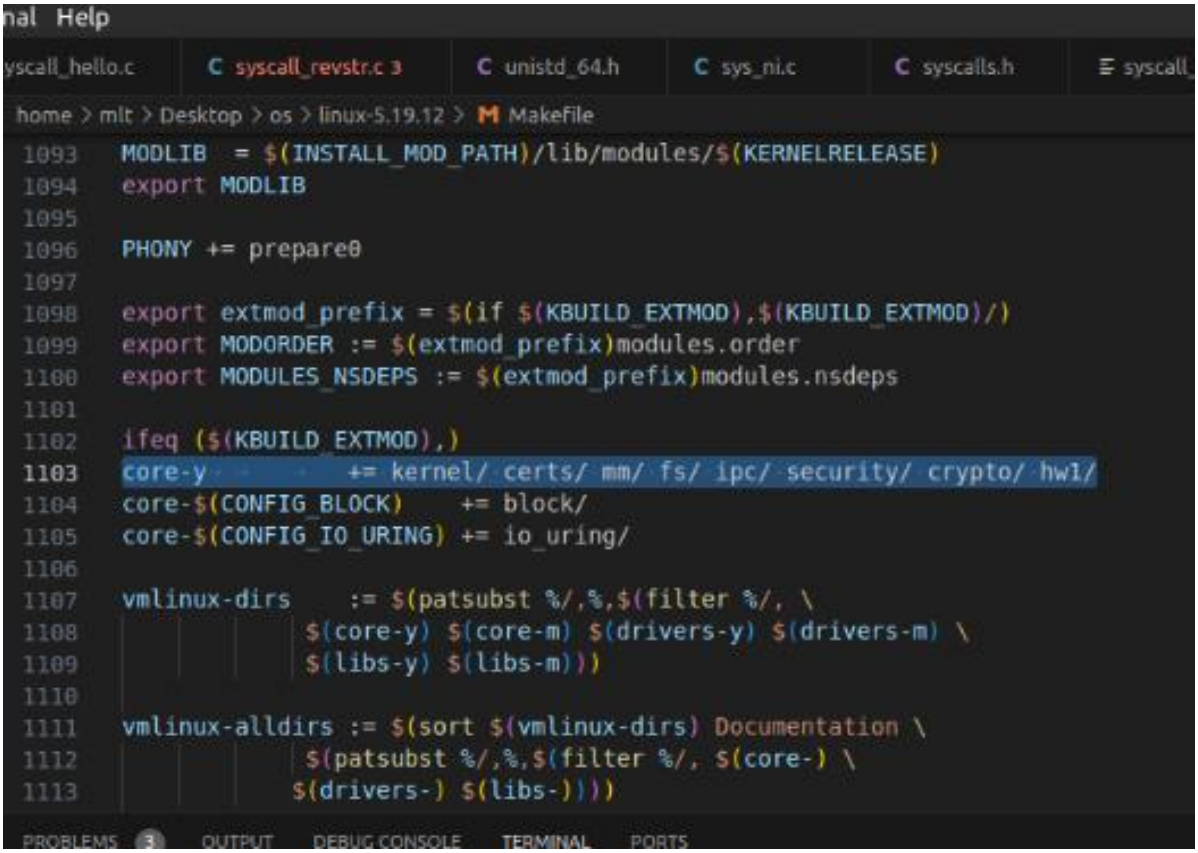
This makefile is put under the same folder as our system call implementations. When the configs of our system call (refer to step 7) is activated, this makefile would compile our system calls into the new kernel.



6. **Modify kernel_path/Makefile:**

Since I put my system call programs in the kernel_path/hw1

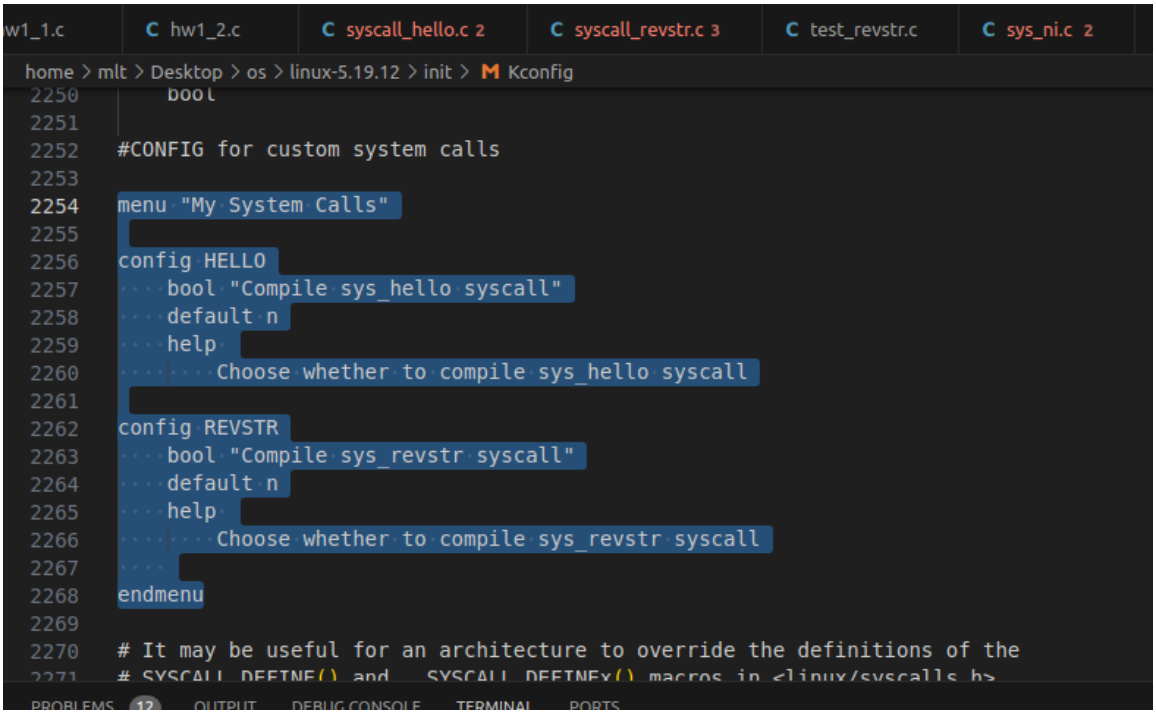
folder(I also tried to put inside the “init” folder, and it worked even without modifying this makefile), so I must add path to the ‘core-y’ in Makefile to let my system call programs can be correctly detected and compiled.

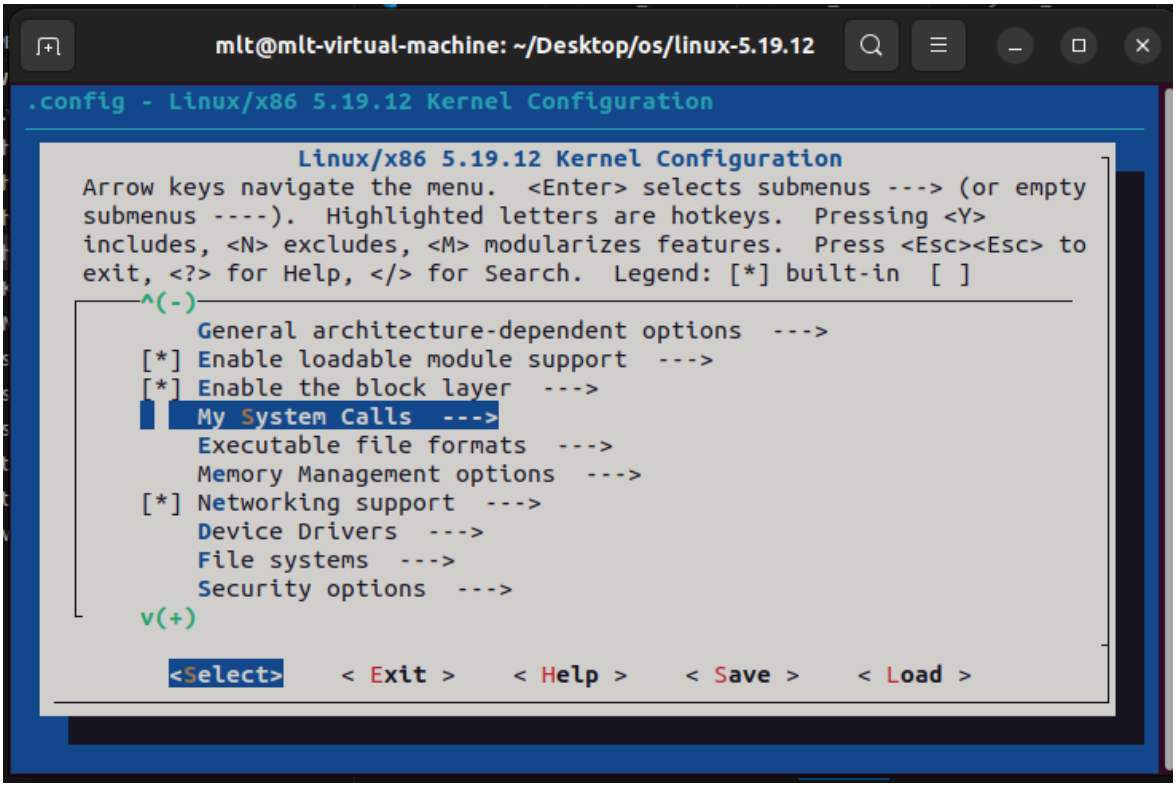


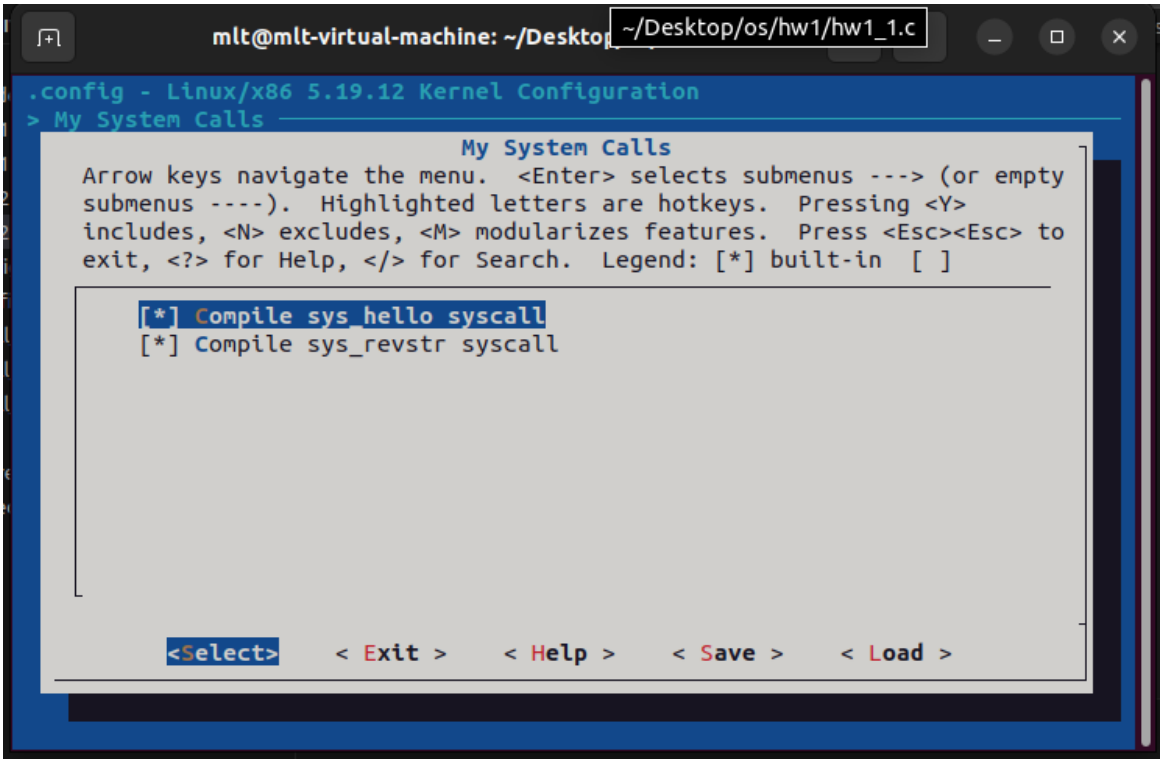
7. Modify kernel_path/init/Kconfig for menuconfig:

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.

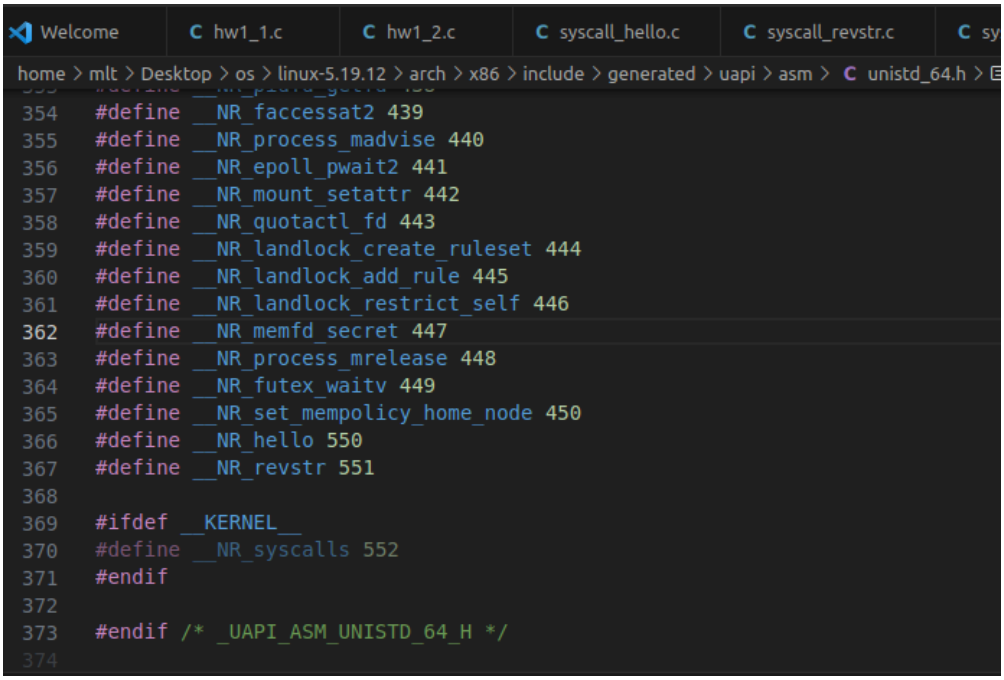

w1_1.c
hw1_2.c
syscall_hello.c 2
syscall_revstr.c 3
test_revstr.c
sys_ni.c 2
home > mlt > Desktop > os > linux-5.19.12 > init > Kconfig
2250 bool
2251
2252 #CONFIG for custom system calls
2253
2254 menu "My System Calls"
2255
2256 config HELLO
2257 bool "Compile sys_hello syscall"
2258 default n
2259 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
PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS


mlt@mlt-virtual-machine: ~/Desktop/os/linux-5.19.12
.config - Linux/x86 5.19.12 Kernel Configuration
Linux/x86 5.19.12 Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]
^(-)
General architecture-dependent options ---->
[*] Enable loadable module support ---->
[*] Enable the block layer ---->
My System Calls ---->
Executable file formats ---->
Memory Management options ---->
[*] Networking support ---->
Device Drivers ---->
File systems ---->
Security options ---->
v(+)
<Select> < Exit > < Help > < Save > < Load >



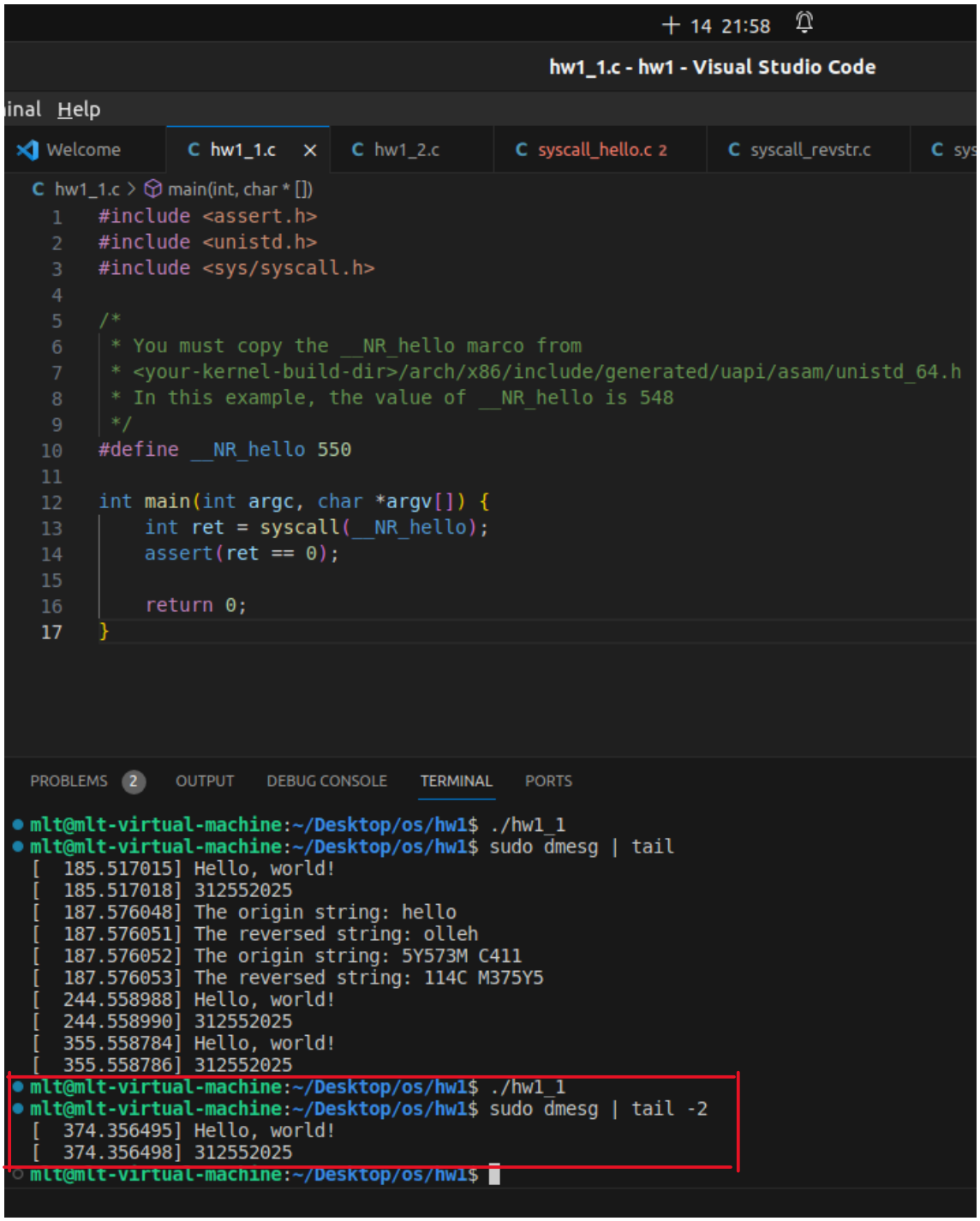
8. **Recompile kernel:**
   (1) First go to the kernel folder(*cd ~/Desktop/OS/linux-5.19.12* in my case)
   (2) type '*make menuconfig*' to activate the options of my system calls as the pictures of last step showed.
   (3) Then type '*sudo make -j8 > /dev/null && sudo make modules_install install && 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.

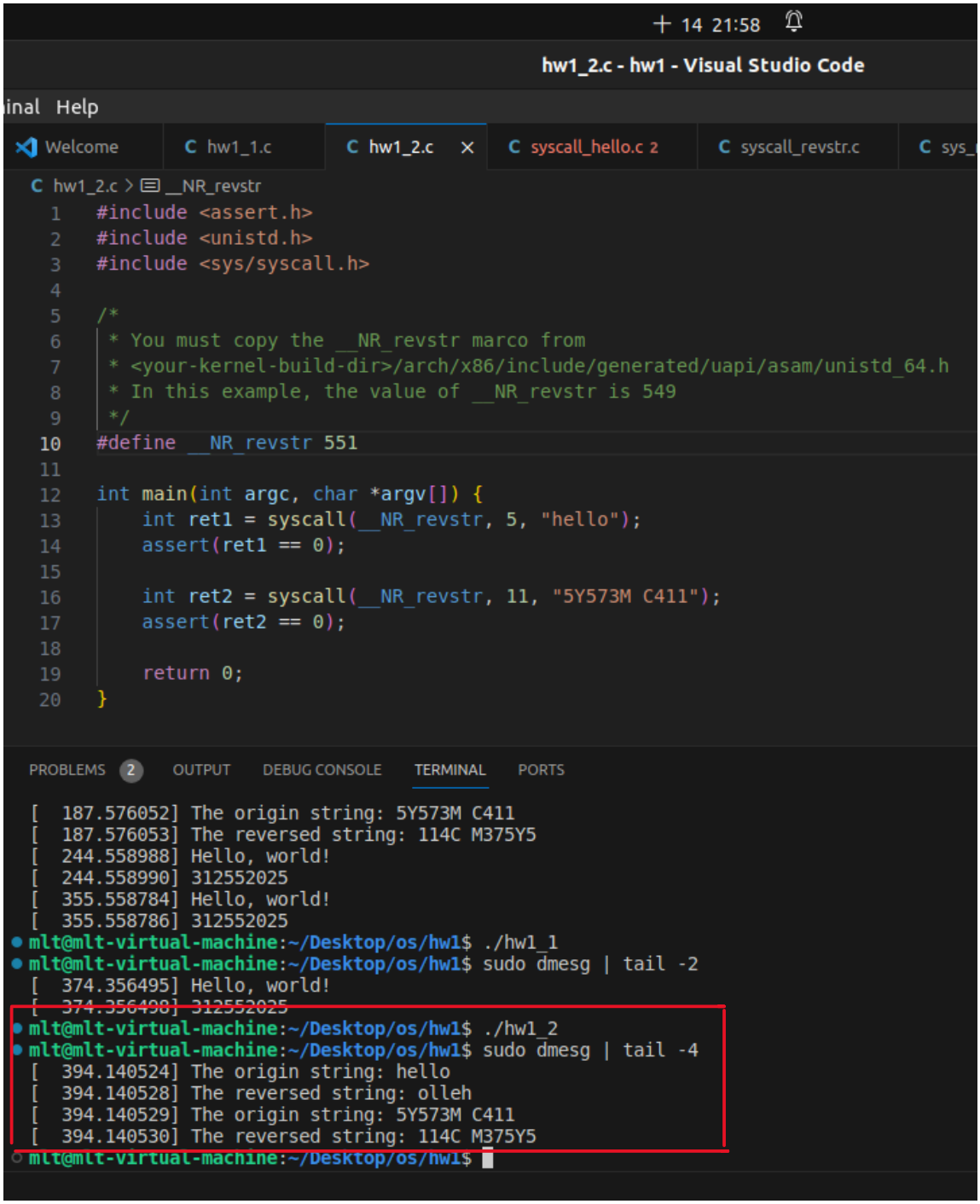


Then compile the C codes provided by TA using gcc, then execute them. To check the outputs of the programs, we must use '*sudo dmesg | tail*' to show the latest log messages.

(1) sys_hello:


14 21:58
hw1_1.c - hw1 - Visual Studio Code
inal Help
Welcome
hw1_1.c
hw1_2.c
syscall_hello.c 2
syscall_revstr.c
hw1_1.c > main(int, char * [])
#include <assert.h>
#include <unistd.h>
#include <sys/syscall.h>
/*
* You must copy the __NR_hello marco from
* <your-kernel-build-dir>/arch/x86/include/generated/uapi/asam/unistd_64.h
* In this example, the value of __NR_hello is 548
*/
#define __NR_hello 550
int main(int argc, char *argv[]) {
int ret = syscall(__NR_hello);
assert(ret == 0);
return 0;
}
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:


14 21:58
hw1_2.c - hw1 - Visual Studio Code
inal Help
Welcome
hw1_1.c
hw1_2.c
syscall_hello.c 2
syscall_revstr.c
hw1_2.c > __NR_revstr
#include <assert.h>
#include <unistd.h>
#include <sys/syscall.h>
/*
* You must copy the __NR_revstr marco from
* <your-kernel-build-dir>/arch/x86/include/generated/uapi/asam/unistd_64.h
* In this example, the value of __NR_revstr is 549
*/
#define __NR_revstr 551
int main(int argc, char *argv[]) {
int ret1 = syscall(__NR_revstr, 5, "hello");
assert(ret1 == 0);
int ret2 = syscall(__NR_revstr, 11, "5Y573M C411");
assert(ret2 == 0);
return 0;
}
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$