Wahaj Khan
19K-1504
Section: J

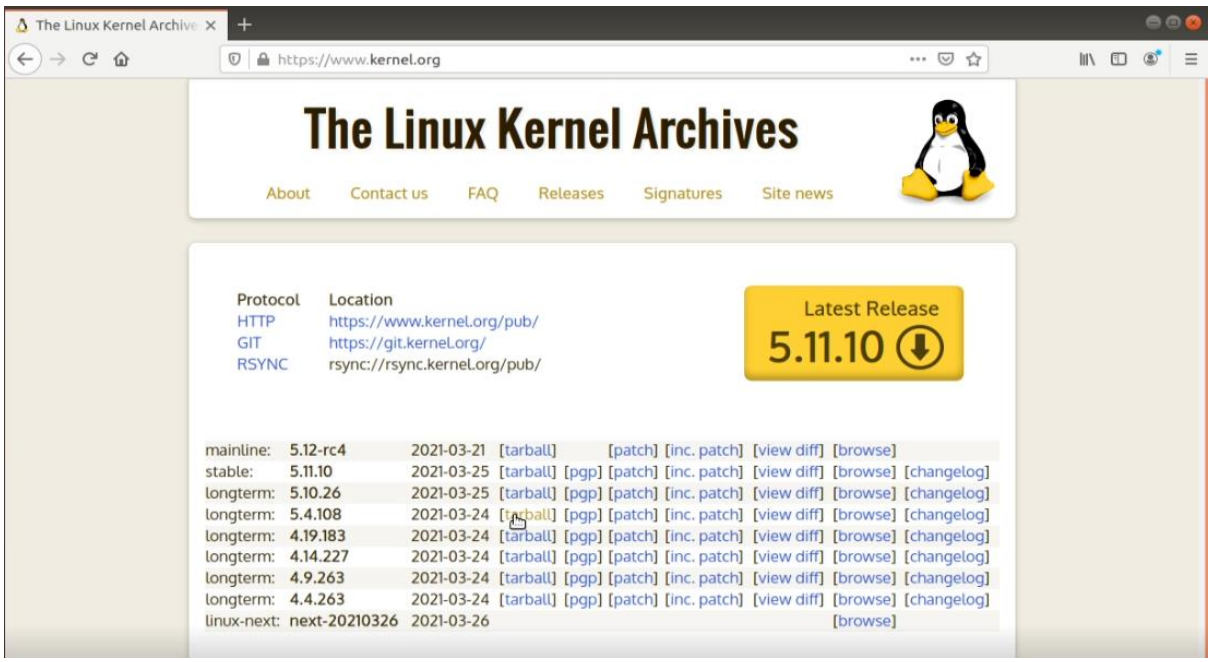
Adding a new System Call to your Kernel

Prerequisites:

- sudo apt-get install gcc
- sudo apt-get install libncurses5-dev
- sudo apt-get install bison
- sudo apt-get install flex
- sudo apt install make
- sudo apt-get install libssl-dev
- sudo apt-get install libelf-dev
- sudo add-apt-repository "deb http://archive.ubuntu.com/ubuntu $(lsb_release -sc) main universe"
- sudo apt-get update
- sudo apt-get upgrade

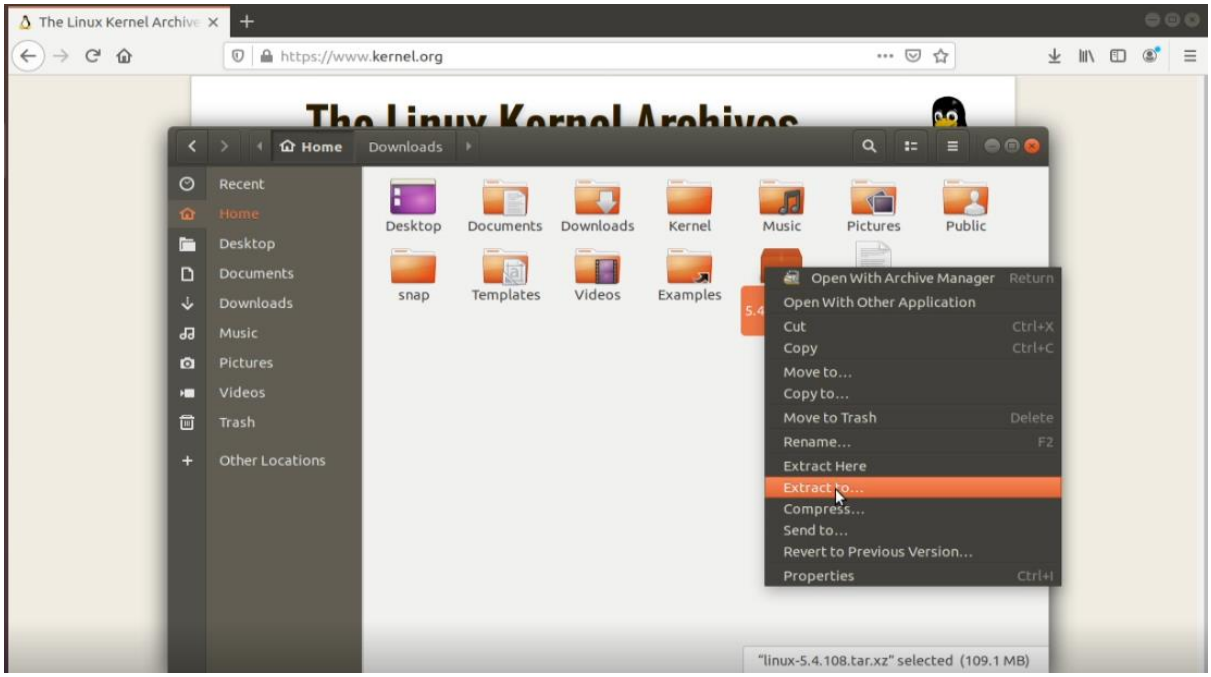
Steps:

1. Downloading a kernel:
   First of all, we have to download a kernel from kernel.org. We can either do this by using the “wget” command or download it manually by clicking the “tarball” option.

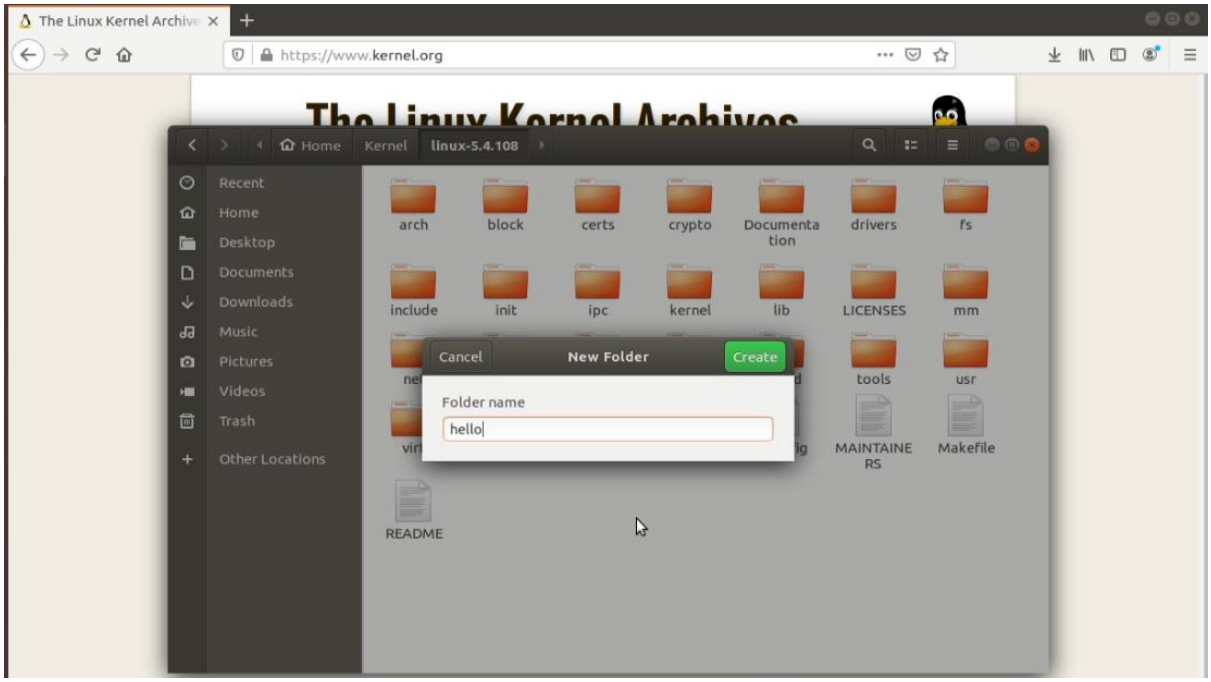


Wahaj Khan
19K-1504 Section: J

2. Extracting the kernel:
   Now, go to the folder where the kernel is downloaded and extract it by typing “tar -xvf *filename*” or simply by right clicking it and selecting the “extract to” option



3. Making a new folder called hello:
   Go into the folder where you extracted the kernel and go inside the kernel’s folder and create a new directory by either opening the terminal there and typing “mkdir *folder name*” or simply by right clicking there and clicking on new folder



4. Adding a C code for the system call:
   Now, go to the folder which we created just now and open the terminal there and create a new C code file by typing “gedit hello.c” and paste the following code there:

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

asmlinkage long sys_hello(void)

{

        printk("Hello world\n");

        return 0;

}
```

Code explanation:

a. We used #include <linux/kernel> because we are building a system call for our linux kernel.
b. Amslinkage simply means that the arguments for this function will be on the stack instead of the CPU registers.
c. Printk is used instead of printf because we are going to print in the kernel's log file.
d. If the code is run and it returns 0, then it will mean that our program ran successfully and Hello world is written to out kernel's log file.

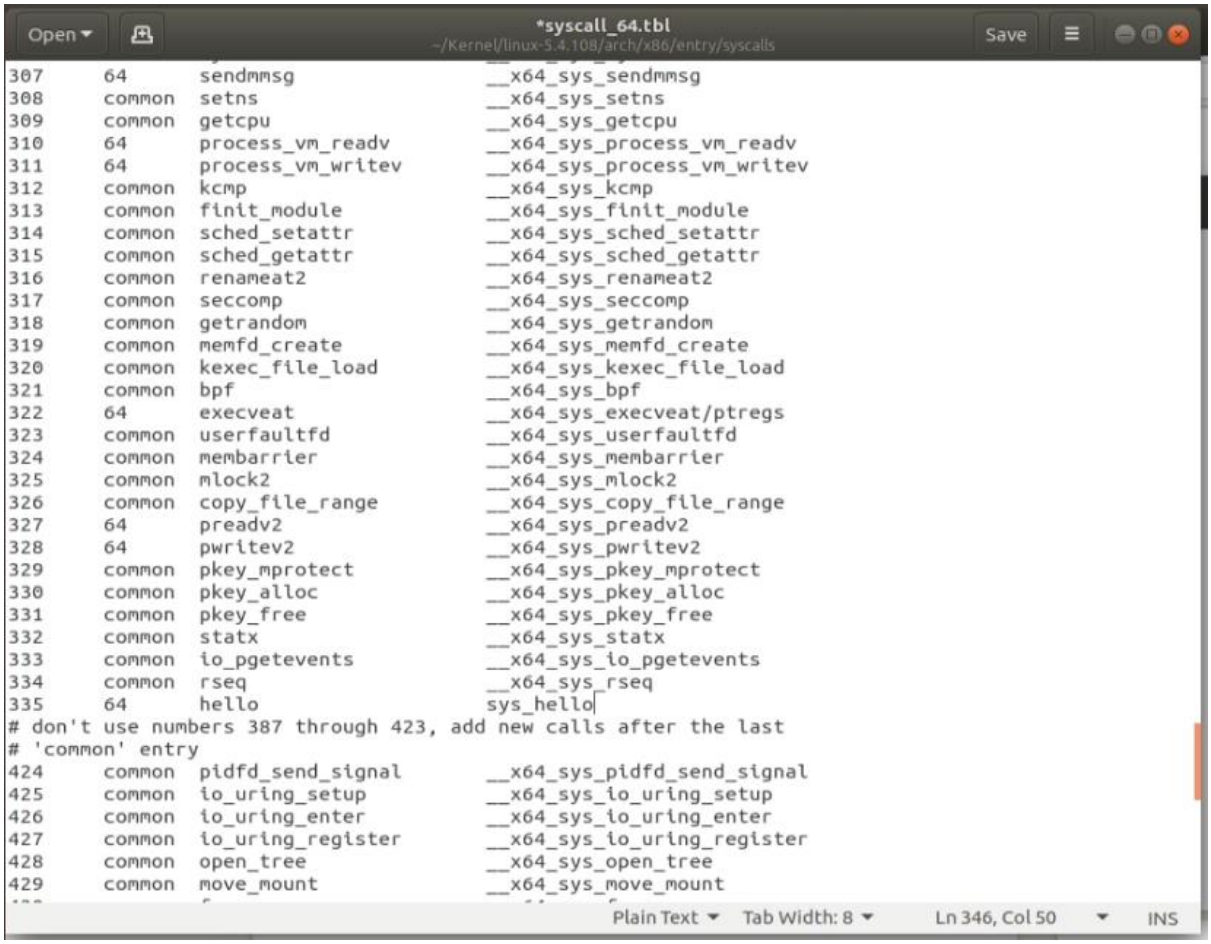
5. Creating a Makefile for the C code:

Now, we have to create a Makefile for our new folder to ensure that the code in the folder is always compiled whenever the kernel is compiled. In order to do this, we type in our terminal "gedit Makefile" and put "obj-y := hello.o"

Wahaj Khan
19K-1504 Section: J

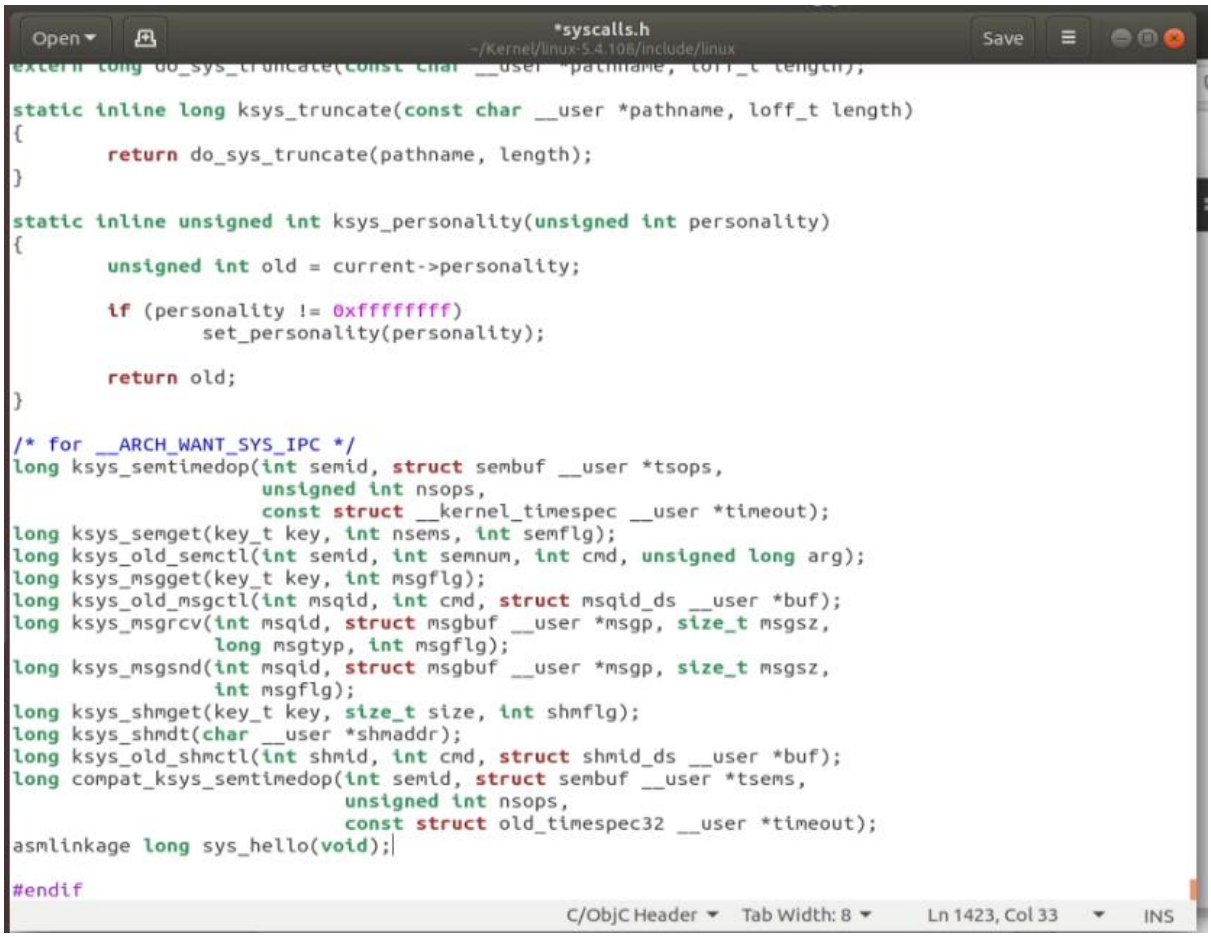
6. Adding the new code into the system table file:

Since we are creating a 64-bit system call according to our system we have to add the system call entry into the syscall_64.tbl file which keeps the name of all the system calls in our system. If our system was a 32-bit system, we would have to add our system call into syscall_32.tbl (We can check the type of our system by typing “uname -m” in a terminal). This tbl file is located inside the kernel folder in /arch/x86/entry/syscalls/syscall_64.tbl. We can go into this directory by using cd and then edit the file by typing “gedit syscall_64.tbl”

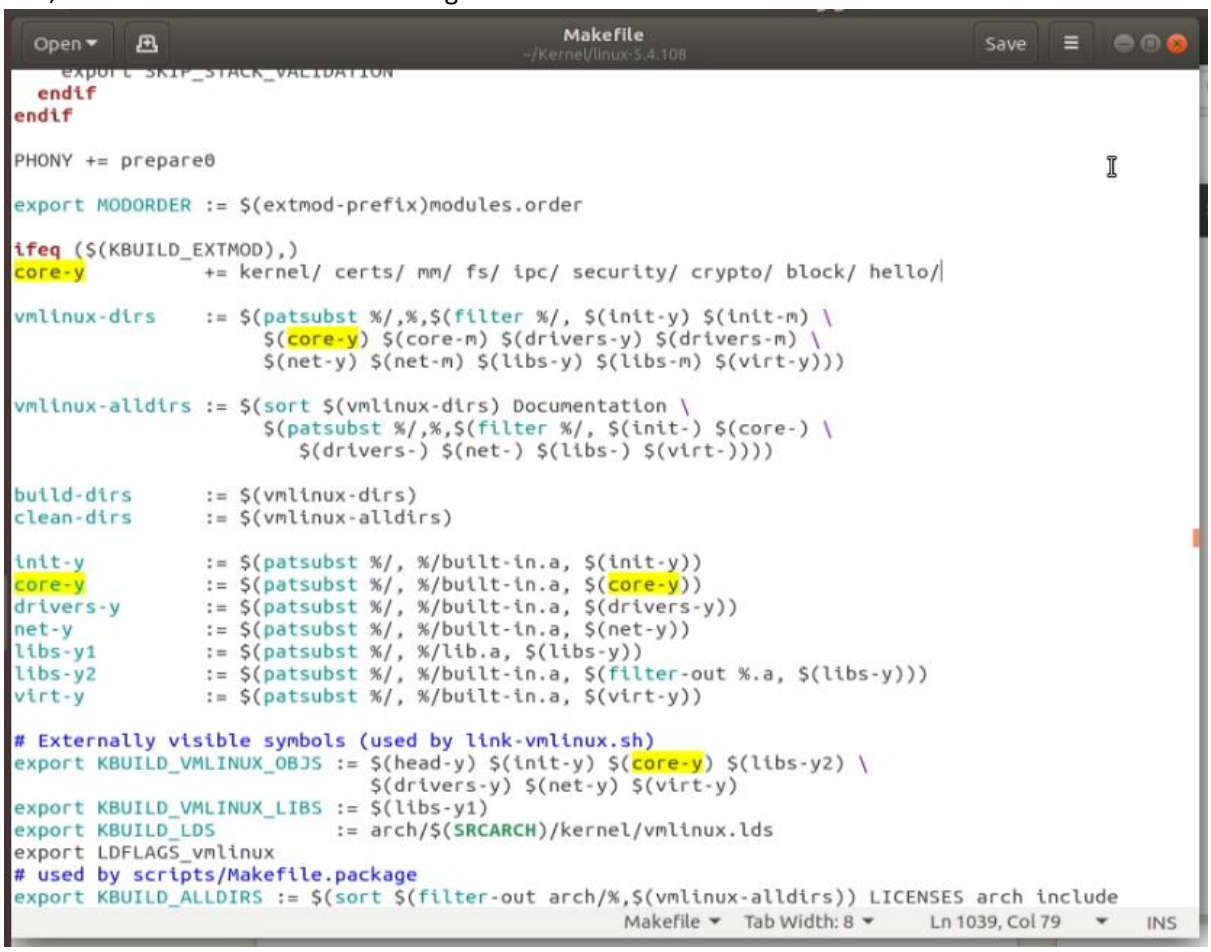


Wahaj Khan
19K-1504 Section: J

7. Adding the prototype of the new system call into the system calls header file:
   Now we have to add the prototype of our system call in the system's header file which is located in the kernel folder then "/include/linux/syscalls.h". We have to add the prototype of our system call function in this file.



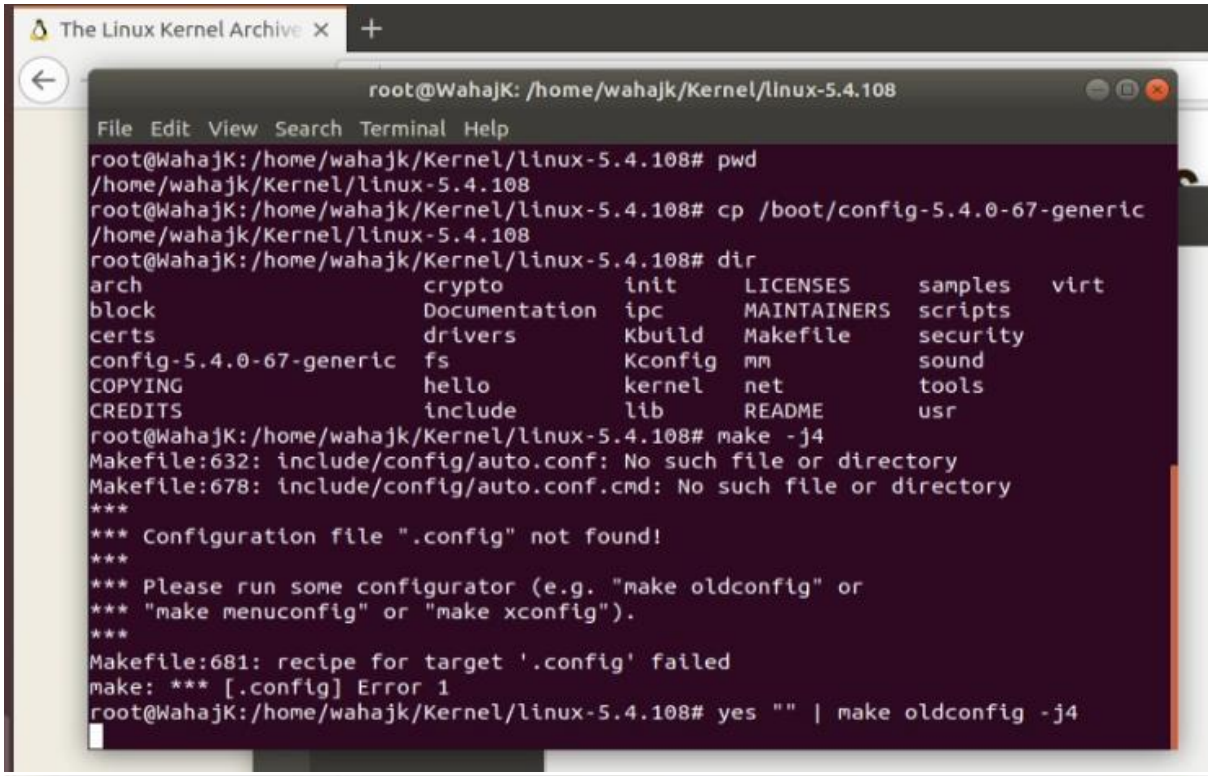
8. Changing version and adding the hello folder in the kernel's Makefile:
   Now, we have to add our roll number in the extraversion of the kernel's make file and we have to add the new module that we created into our kernel's make file. For this, we open the Makefile of the kernel and search for "core-y" and go to it's second instance which is under "KBUILD_EXTMOD" and add our new module which is "hello" at the end of it. At the end, our make file will look something like this:

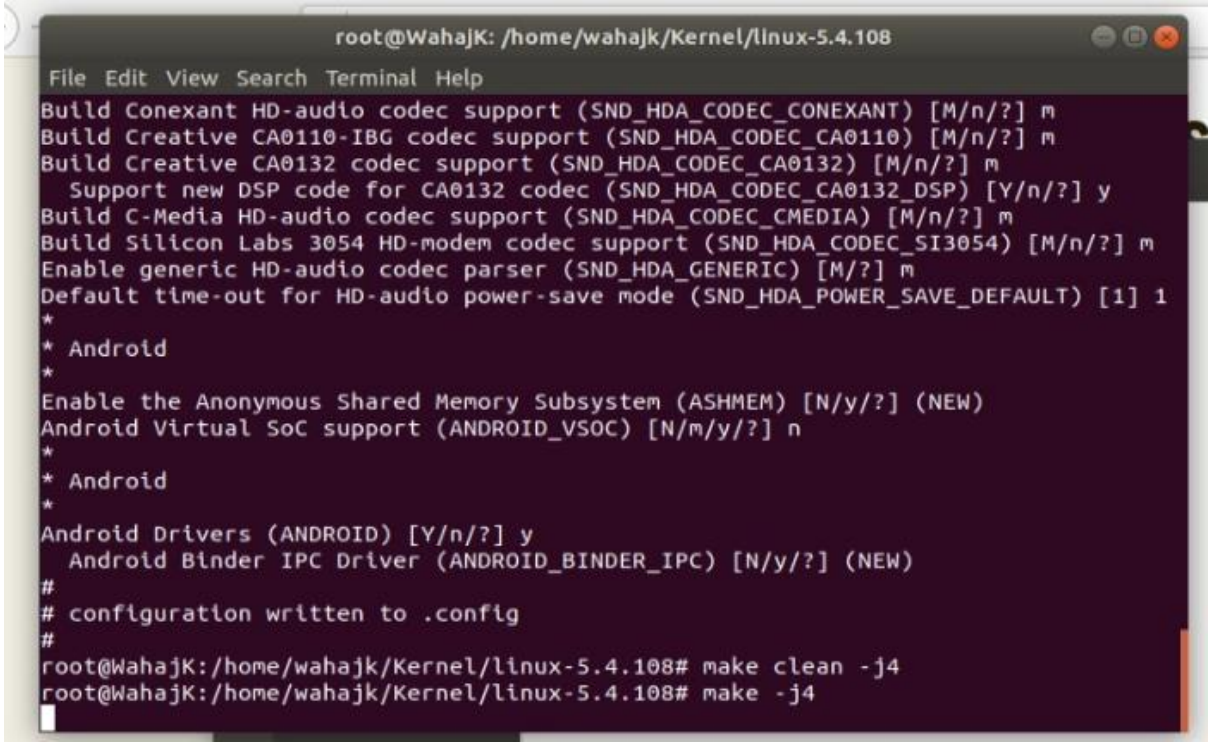


9. Creating a config file:
   Now we have to create a config file for our kernel. The order of the steps before this can change but the order of this step and the steps coming right after it can't change. We can either create a Menuconfig or simply copy the oldconfig. I will be copying the oldconfig and using that config for my new kernel. First of all, we search for the config that we currently have by typing "ls /boot | grep config" and the we copy the config that is shown to us by typing "cp /boot/config-4.10.0-28-generic *our linux kernel directory*". Then we create the old config by typing "yes "" | make oldconfig -j4", by doing so, the system will automatically create the new config for us and select the default option for everything.

10. Cleaning and Compiling the kernel:
    We have to clean all of our existing object and executable file because compiler sometimes link or compile files incorrectly and to avoid this, we delete all of our old object and executable files by typing "make clean -j4" (It is better to switch to super user mode by typing "sudo su" before running the commands after this) and when this all is done, we type "make -j4" to start building our kernel (-j4 allocates the multiple cores that our system have for compiling. If we don't do this, the system will only use a single core for compiling the

kernel which is rather slow. Adding -j4 will increase the speed of our compiling by almost 4 times. Note that 4 is the number of Cores that I have in my Laptop. You can check your number of cores by typing the command “lscpu” and enter the number accordingly)





Now we have to wait until our Kernel image is built and ready. If we see “Kernel image is ready” when the command is done executing, that means that our kernel image is ready to be installed. (I forgot to take screenshots of this point)

Wahaj Khan
19K-1504 Section: J

11. Installing modules:

Now we have to install the kernel that we built by typing “make modules_install install” which will install the kernel and update our grub as well. When this all is done and the terminal says “done”, then we can restart our laptop either manually or by typing “shutdown -r now” and hold the “Shift” key while it is restarting to open up the grub menu and switch to the new kernel which we just installed. (I forgot to take screenshot of this point as well.)

12. Checking if the System call is Working Properly:

After logging into the newly compiled kernel, we check the system call by making a C code named “userspace.c” and putting the following code in it:

```
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>
int main()
{
	long int i = syscall(335);
	printf("System call sys_hello returned %ld\n", i);
	return 0;
}
```

Now we compile the code by typing “gcc userspace.c” and executing it by typing “./a.out”. If it returns 0, this means that our code has compiled successfully and the system call is working fine (Note that in calling syscall(335), 335 is the number where we added our system call in the table) and finally, we run “dmesg” to see the kernel messages and we will find “Hello World” written at the end of it.