Reading Material for System Call Implementation in Linux Operating System [Including Kernel Recompilation]

As Part of

Operating Systems [CS F372] Course

Semester I, 2020 – 2021

BITS Pilani
K K Birla Goa Campus

Please contact: [only if you fail miserably!!]

CHIPLUNKAR CHINMAY VIDYADHAR [2017A7PS0097G] and
SHASHWAT BADONI [2017A7PS0115G]

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI –K K BIRLA GOA CAMPUS

INDEX

Important Note: Please **do not try this directly on your laptop**, you may end up crashing your system/corrupting several files. We recommend installing Virtual Box and setting up a Linux VM on it. You can find instructions on how to do that here: https://itsfoss.com/install-linux-in-virtualbox/. Proceed with the next steps only after you a Ubuntu VM up and running in Virtual Box.

Basic Preparation for Recompilation

Step #1. Download the kernel source 5.8.8 from https://www.kernel.org

Step #2: Open a terminal and login to super-user by

$ **sudo su**
<Enter root password here>

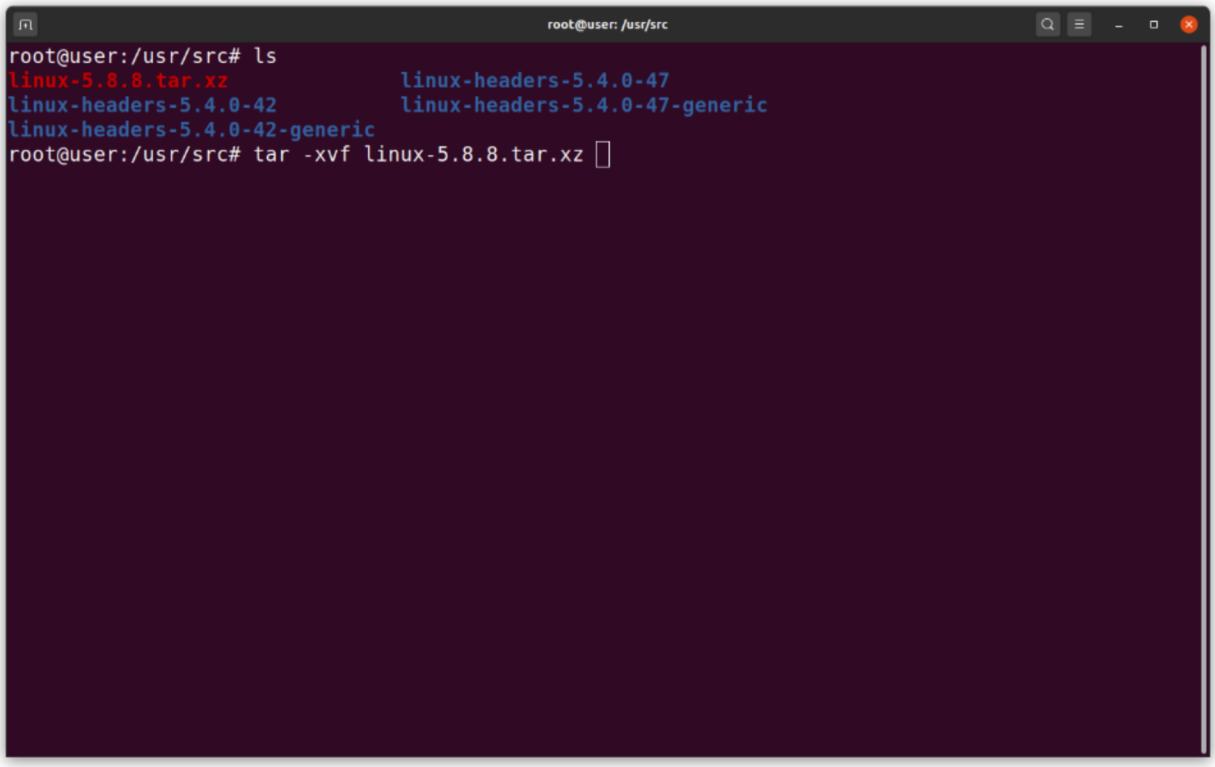
Step #3: Place the tar.xz file in /usr/src/ directory

Step #4: Set the present working directory as /usr/src/ by

$ ***cd /usr/src/***

Step #5: Untar the ***linux-5.8.8.tar.xz*** file by

$ ***tar -xvf linux-5.8.8.tar.xz***



Step #6: Set the present working directory as linux-5.8.8

$ ***cd linux-5.8.8/***

Recompilation of Linux Kernel with / without Modification(s)

Step #1: Reconfiguration of the Kernel

The Linux Kernel is extraordinarily configurable; you can enable and disable many of its features, as well as set build parameters.
Some of the widely used options are: menuconfig, xconfig, gconfig, oldconfig, defconfig etc.

Dependencies you may require to install: flex, bison, libssl-dev, libelf-dev

$apt install flex bison libssl-dev libelf-dev or

$apt-get install <package name> E.g.: $apt-get install flex

$ make menuconfig

<Text based color menus, radio lists & dialogs. This option is also useful on remote server if you want to compile kernel remotely.>

$ make xconfig

<X windows (Qt) based configuration tool, works best under KDE Desktop.>

$ make gconfig

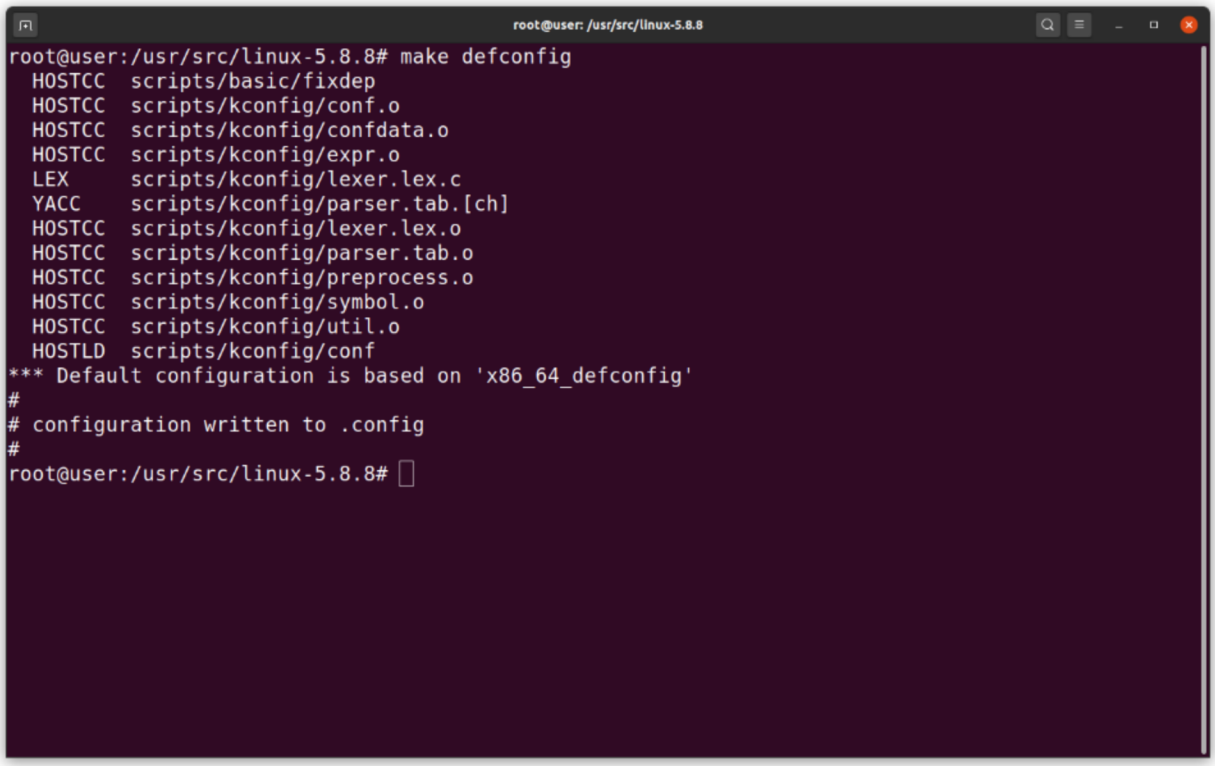
<X windows (Gtk) based configuration tool, works best under Gnome Desktop.>

$ make oldconfig

<Reads the existing config file and prompts the user options in the current kernel source that are not found in the file>

$ make defconfig [Use this for reconfiguration option for this assignment]

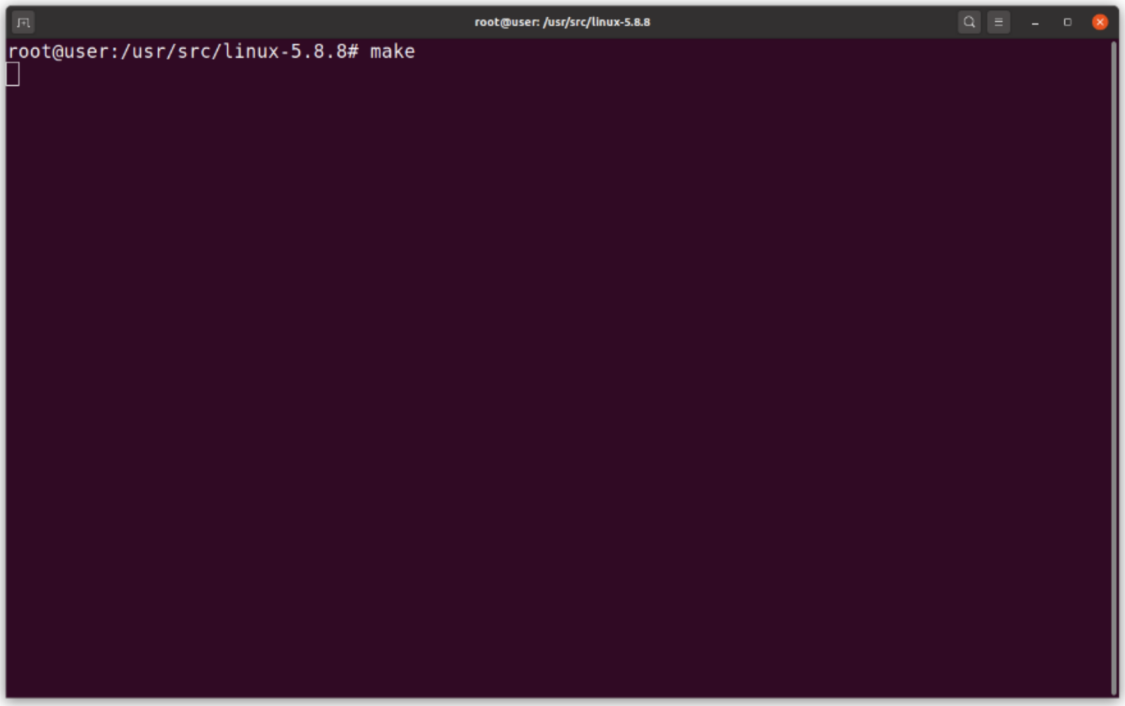
<Creates a default config file for the kernel delineating all the necessary modules to be installed into the kernel>

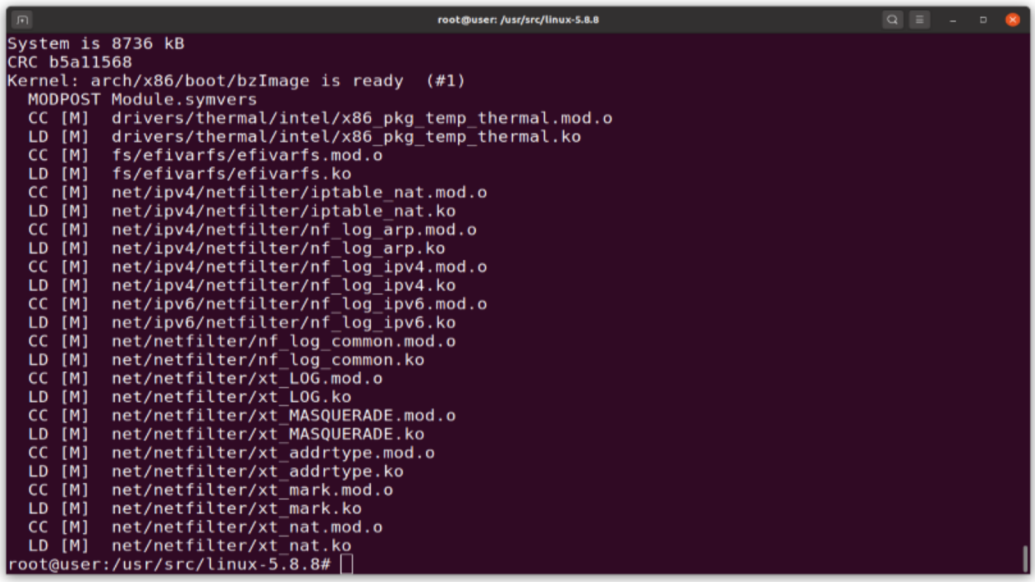


Step #2: Preliminary Recompilation of the Kernel

Execute make to compile the kernel.

$ make





Step #3: Recompilation of the Kernel module, update initramfs and grub

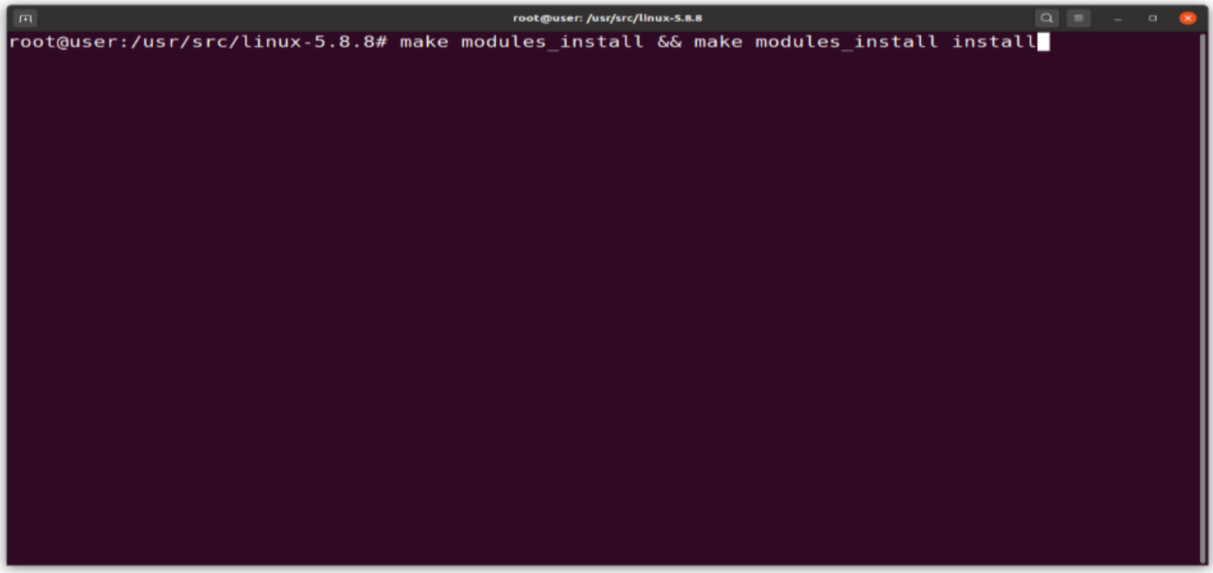
Execute make modules_install & make modules_install install to compile the modules and update the initramfs and grup.

$ make modules_install && make modules_install install

In addition to installing the bzImage it even runs the following commands

update-initramfs -c -k linux-5.8.8

update-grub



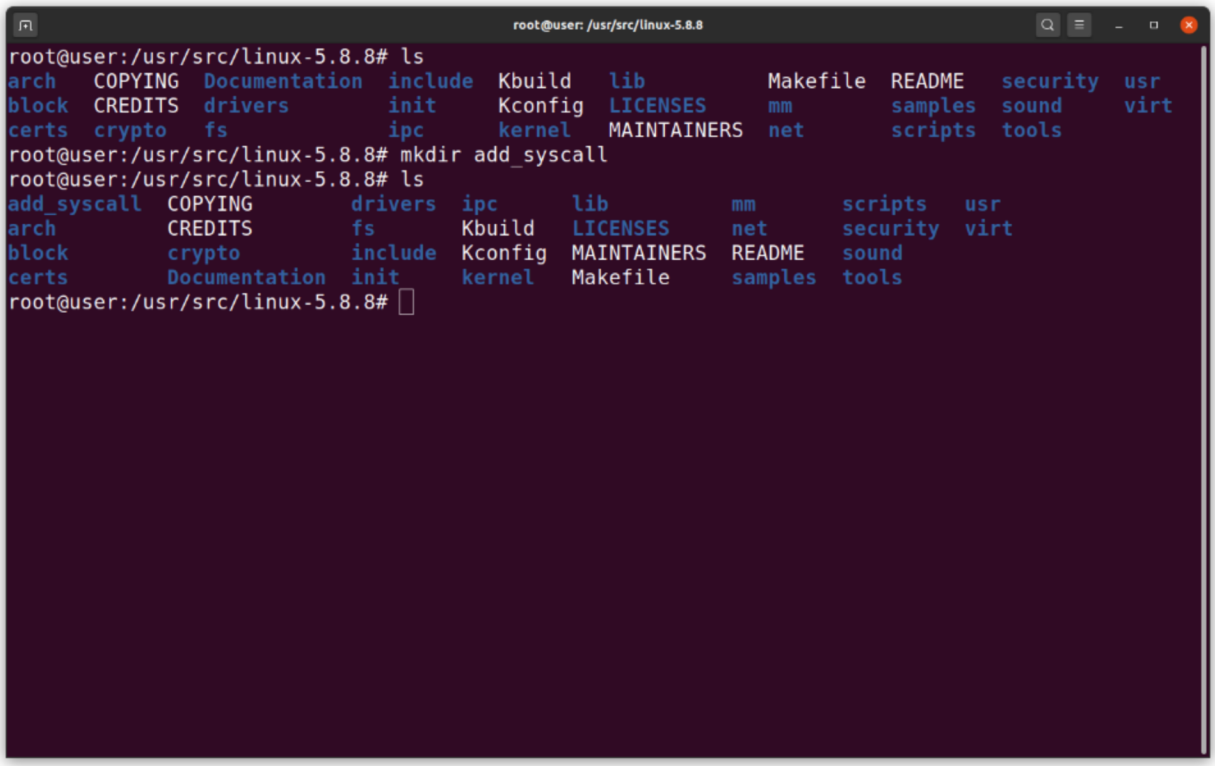
Now that the kernel has been recompiled, reboot the system and boot into this kernel from the grub <Select advanced ubuntu tab followed by the New kernel>

Implementation of New System Call [New System Call to Add 2 Positive Integers]

Step #1: Create a directory under /usr/src/linux-5.8.8/

Create a directory named add_syscall under /usr/src/linux-5.8.8/

$ mkdir add_syscall



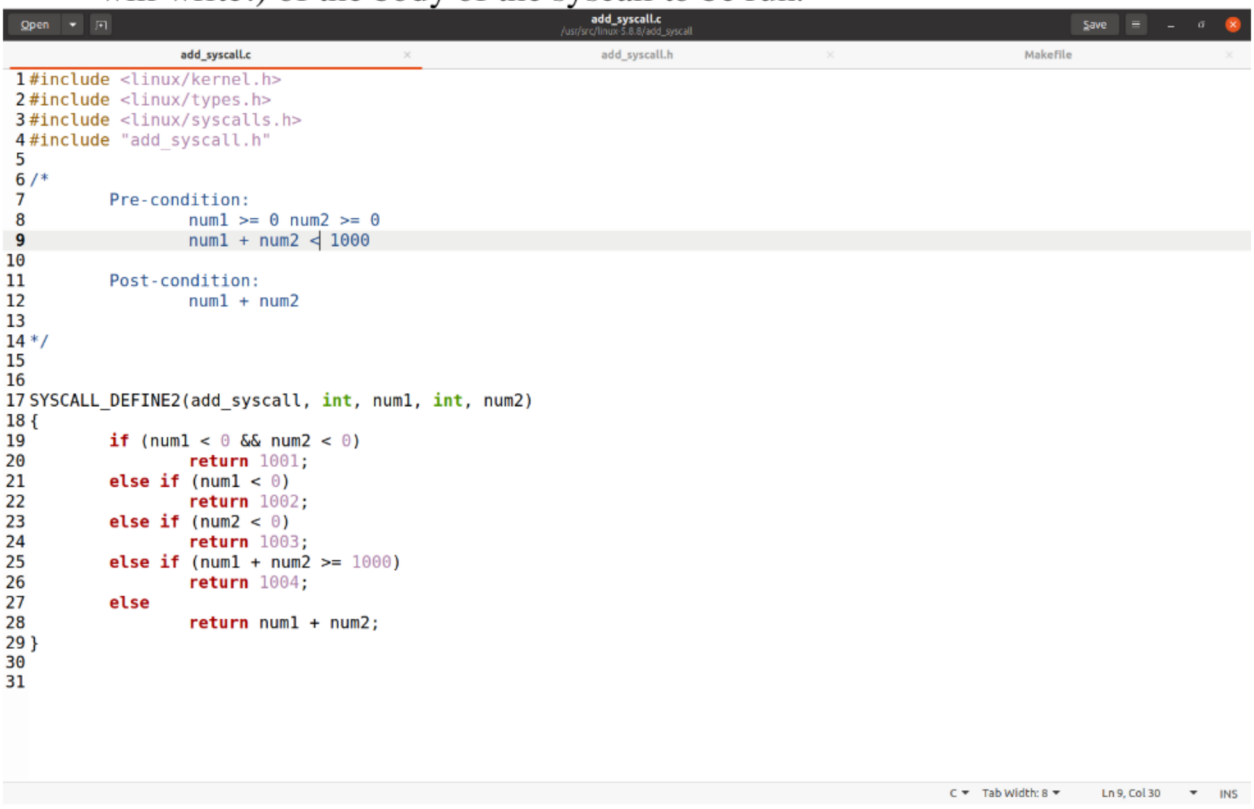
Step #2: Create the following files under add_syscall directory

1. **add_syscall.c**
2. **add_syscall.h**
3. **Makefile**

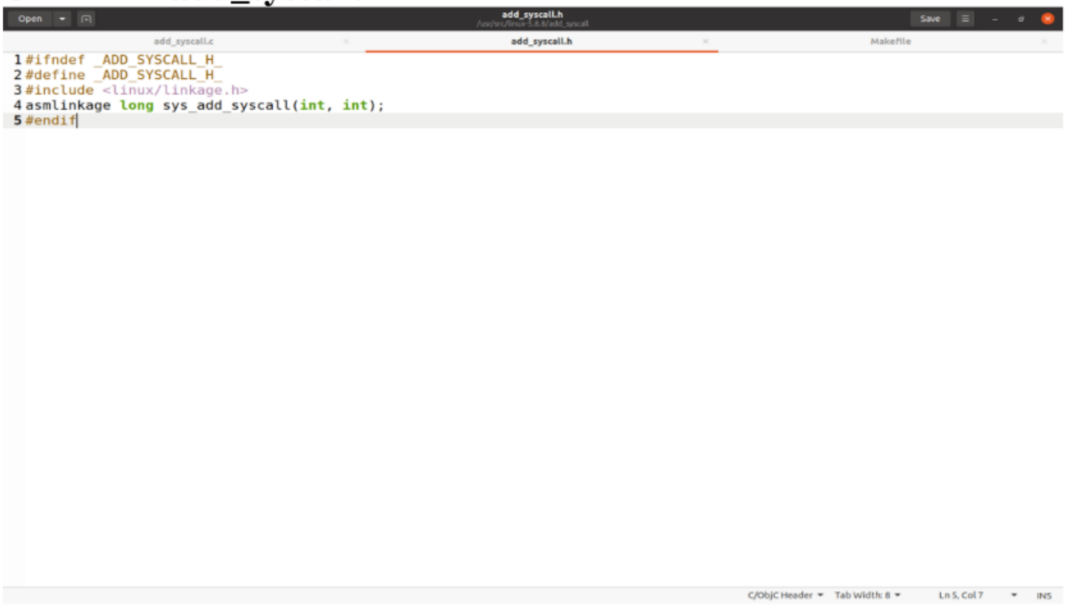
Contents of **add_syscall.c**

SYSCALL_DEFINE*n*() macros are the standard way for kernel code to define a system call, where the *n* suffix indicates the argument count.
The first argument to the macro is the name of the system call (without sys_ prepended to it). The remaining arguments are pairs of type and name for the parameters.

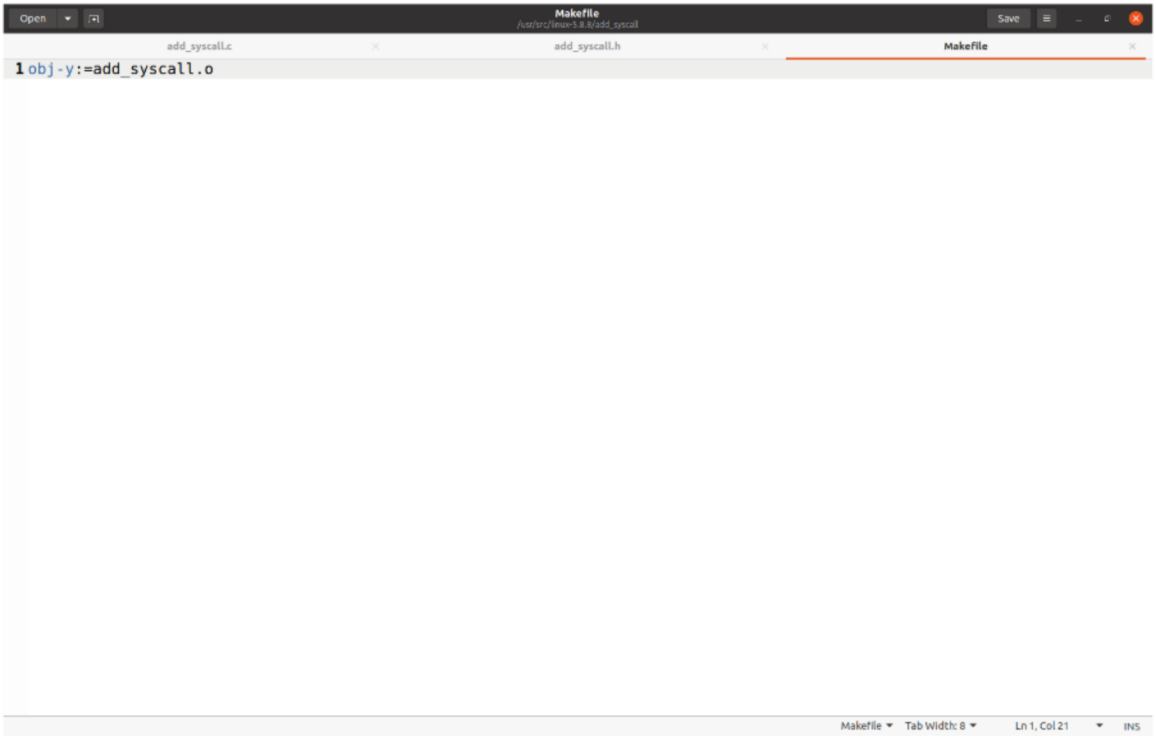
The definitions of these **SYSCALL_DEFINE**... macros are in **#include </linux/syscalls.h>**. Hence, the .c file in which you code the body of your syscall's service routine must **#include <linux/syscalls.h>**
It has within { ... } (after your SYSCALL_DEFINE...(...)) the code (you will write!) of the body of the syscall to be run.



Content of **add_syscall.h**



Content of **Makefile**



Step #3: Modify the following files

1. **/usr/src/linux-5.8.8/Makefile**
2. **/usr/src/linux-5.8.8/arch/x86/entry/syscalls/syscall_64.tbl**
3. **/usr/src/linux-5.8.8/include/asm-generic/syscalls.h**
4. **/usr/src/linux-5.8.8/include/linux/syscalls.h**

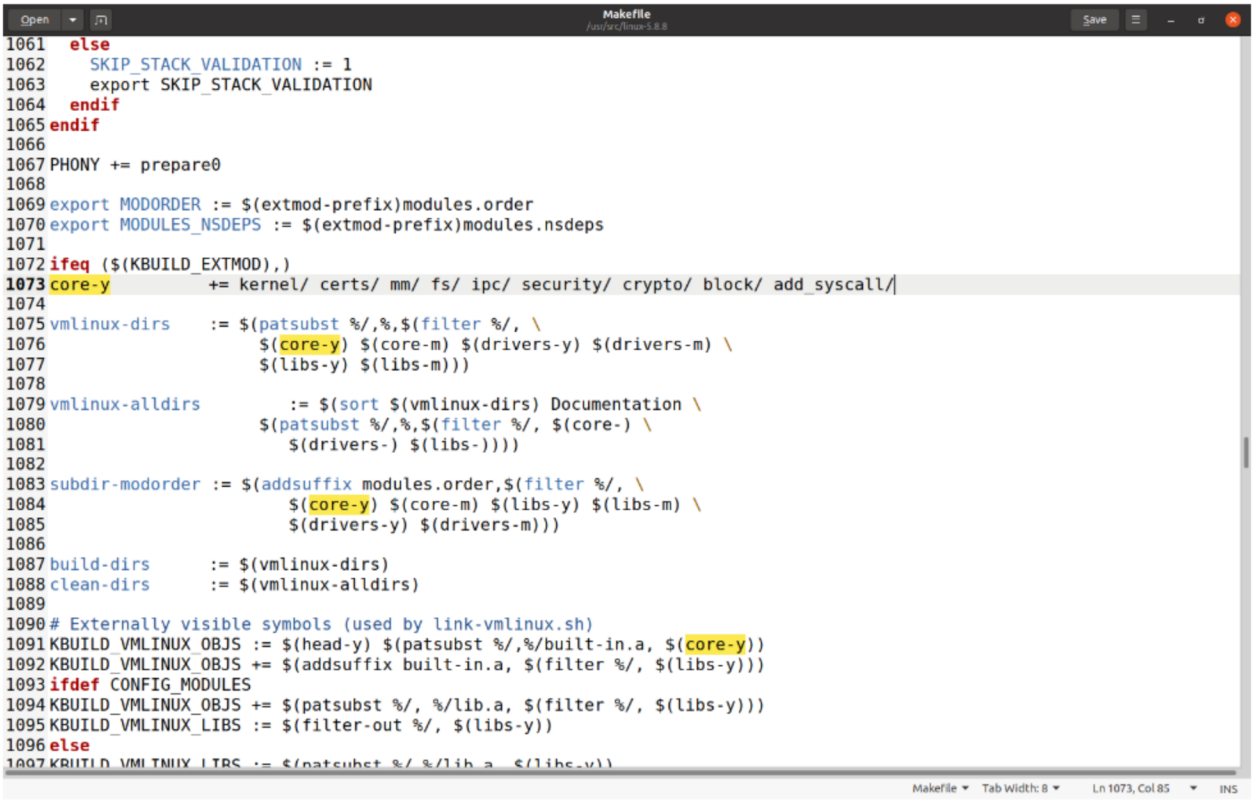
3.1: Modify /usr/src/linux-5.8.8/Makefile:

<Update the following line in Makefile>

core-y += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/

<to the following by adding add_syscall/ in the end>

core-y += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ **add_syscall/**

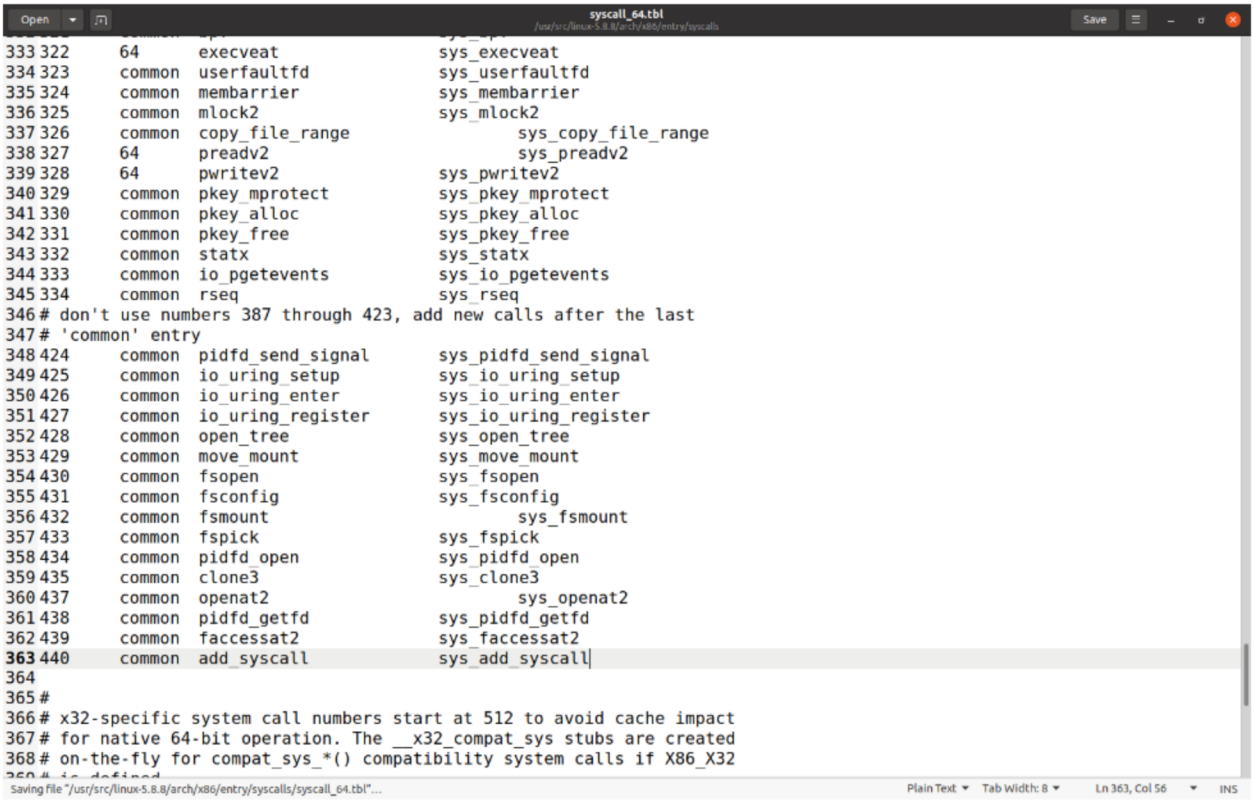


3.2: Modify /usr/src/linux-5.8.8/arch/x86/entry/syscalls/syscall_64.tbl:

Update the file: **/arch/x86/entry/syscalls/syscall_64.tbl** to add the new syscall at the **next available system call number** in the common list of syscalls like:

440 common add_syscall sys_add_syscall

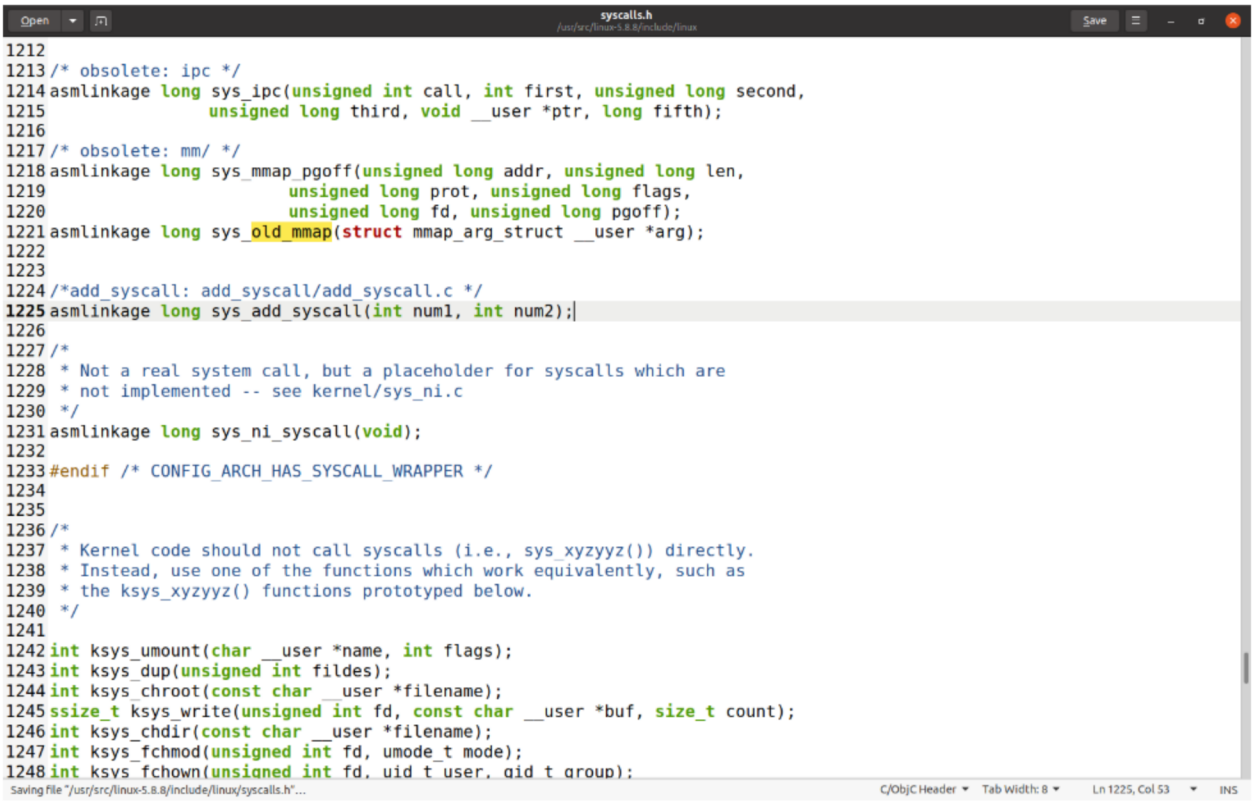
Here sys_add_syscall is the entry point for the system call add_syscall and it will be common across the x86-{64, 32} bit architectures.



This table is read by scripts and used to generate some of the boilerplate code

3.3: Modify /usr/src/linux-5.8.8/include/asm-generic/syscalls.h:

3.4: Modify /usr/src/linux-5.8.8/include/linux/syscalls.h:



Recompile the Kernel [Follow section#2]to get all the changes reflected. Reboot the system and boot into this kernel from the grub <Select advanced ubuntu tab followed by the New kernel>

Implementation of User Space Programs

1. **add2Num.c**
2. **addWrapper.h**

The C user library wraps most system calls for us. This avoids triggering interrupts directly. The user space .c file provides two mechanisms of calling a system call (A) directly using the ***syscall()*** function with the help of system call number [GNU C library provides this for us] and (B) with the help of a Wrapper where the end user never need to remember the system call number.

1.1: add2Num.c:

shashwat@user: ~

```
#include <stdio.h>
#include <unistd.h>
#include "addWrapper.h"
#define MY_SYSCALL 440

int main(int argc, char * arv[])
{
        int num1, num2, res_direct, res_wrapper;
        printf("Enter number 1 and number 2:\n");
        scanf("%d %d", &num1, &num2);
        res_direct = syscall(MY_SYSCALL, num1, num2);
        res_wrapper = add_syscall(num1, num2);

        printf("The result of adding %d with %d\n", num1, num2);
        printf("Result of Direct call is %d\n", res_direct);
        printf("Result of wrapper call is %d\n", res_wrapper);

        return 0;
}
```

1.2: addWrapper.h:

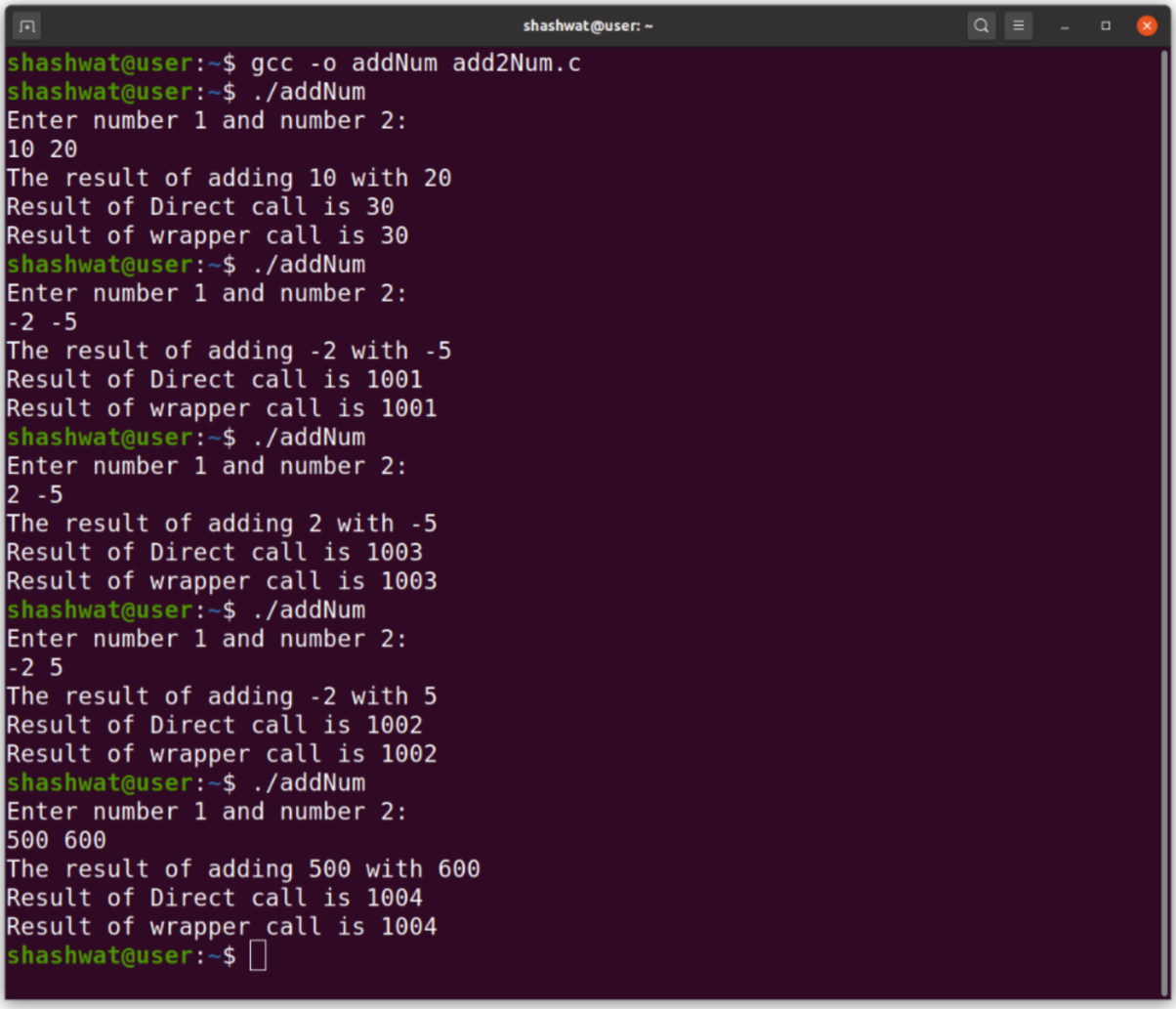
shashwat@user: ~

```
#ifndef __ADD_WRAPPER_H_
#define __ADD_WRAPPER_H_

#define add_syscall(num1, num2) (syscall(440, num1, num2))

#endif
```

1.3: Compiling and Executing the User program:



Practice Problem:

Write a New system call in Kernel space which will add 2 floating point numbers and return the result to the user space. Make sure both the floating point numbers are Valid Positive Numbers. Make sure the result is a Valid Positive floating point number.