

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

Sec. No.	Section Title	Page No.
1.	Basic Preparation for Recompilation	3
2.	Recompilation of Linux Kernel with / without Modification(s)	4
3.	Implementation of New System Call [New System Call to Add Two Positive Integers]	7
4.	Implementation of User Space Programs	13
5.	Practice Problem	14

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>
```

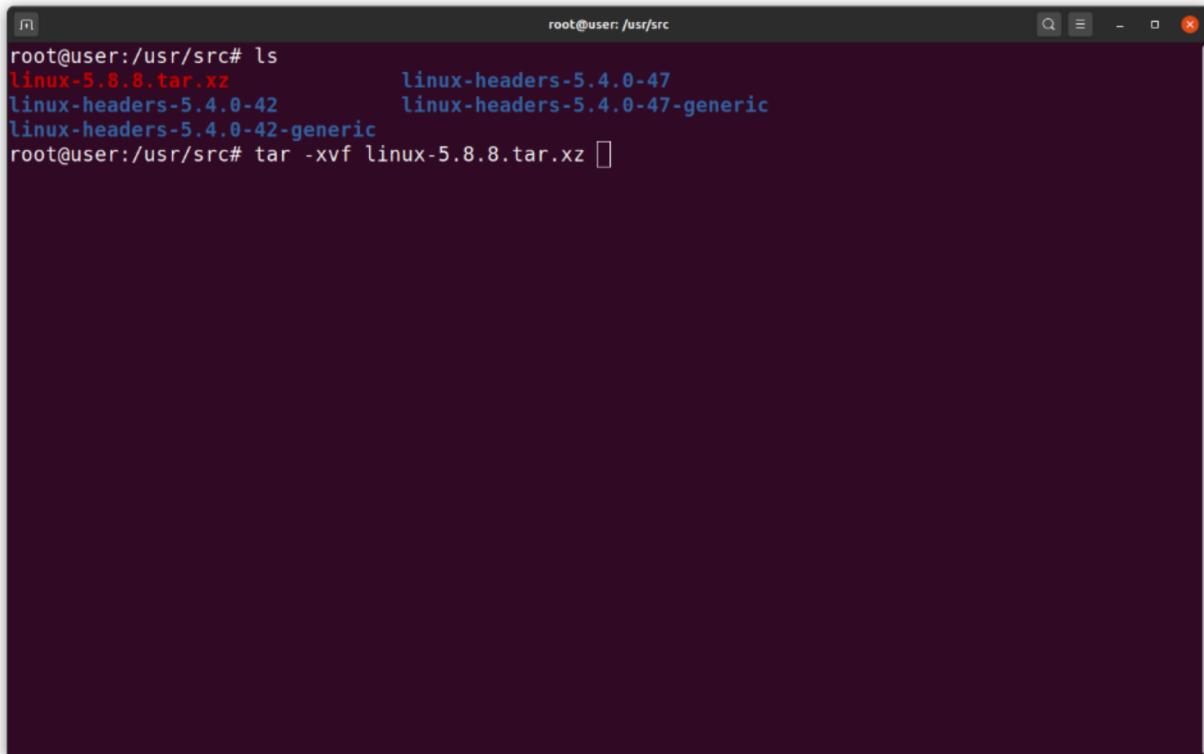
Step #3: Place the tar.xz file in /usr/src/ directory

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
```



The screenshot shows a terminal window with a dark background and light-colored text. The title bar indicates the session is running as 'root' on the host 'user'. The command 'ls' is run first, showing several files: 'linux-5.8.8.tar.xz', 'linux-headers-5.4.0-47', 'linux-headers-5.4.0-42', and 'linux-headers-5.4.0-42-generic'. Then, the command 'tar -xvf linux-5.8.8.tar.xz' is run, which starts extracting the contents of the tar archive.

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

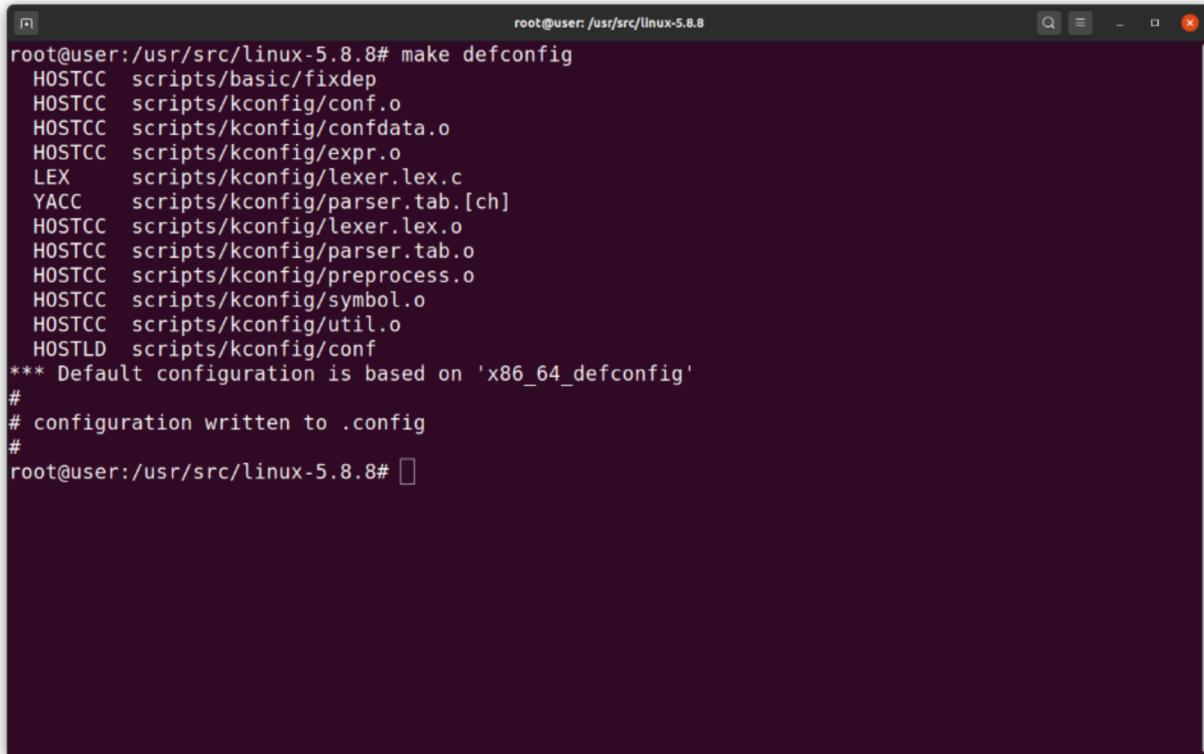
<X windows (Gtk) based configuration tool, works best under Gnome Desktop.>

\$ 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>

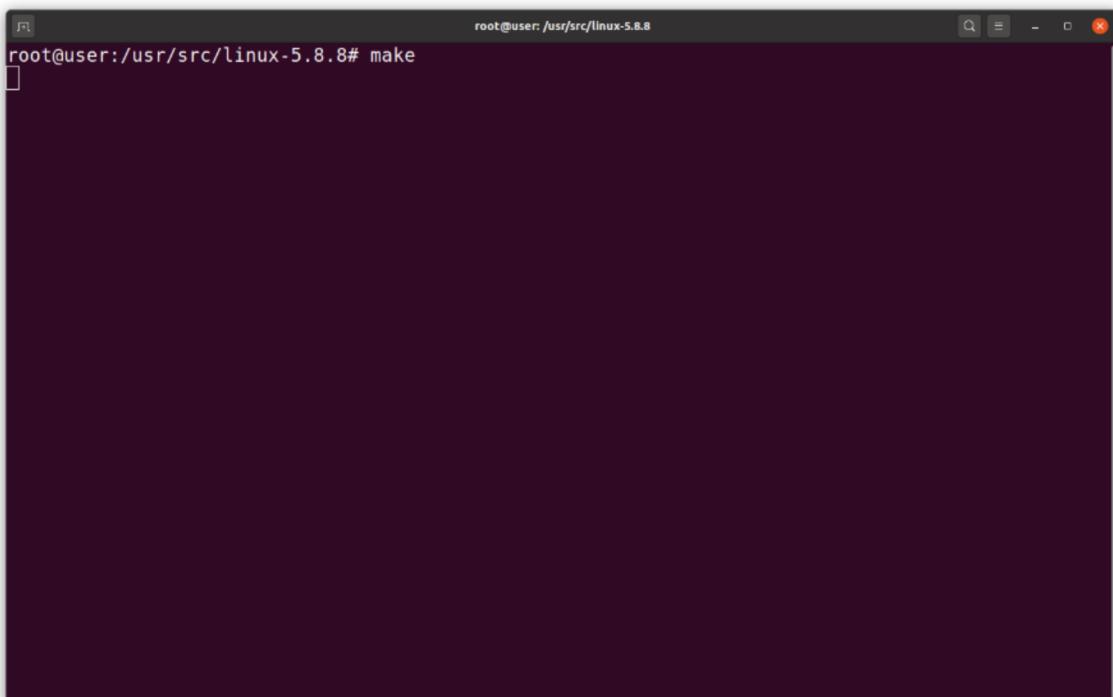


```
root@user:/usr/src/linux-5.8.8# make defconfig
HOSTCC scripts/basic/fixdep
HOSTCC scripts/kconfig/conf.o
HOSTCC scripts/kconfig/confd.o
HOSTCC scripts/kconfig/expr.o
LEX scripts/kconfig/lexer.lex.c
YACC scripts/kconfig/parser.tab.[ch]
HOSTCC scripts/kconfig/lexer.lex.o
HOSTCC scripts/kconfig/parser.tab.o
HOSTCC scripts/kconfig/preprocess.o
HOSTCC scripts/kconfig/symbol.o
HOSTCC scripts/kconfig/util.o
HOSTLD scripts/kconfig/conf
*** Default configuration is based on 'x86_64_defconfig'
#
# configuration written to .config
#
root@user:/usr/src/linux-5.8.8#
```

Step #2: Preliminary Recompilation of the Kernel

Execute make to compile the kernel.

\$ make



```
root@user:/usr/src/linux-5.8.8# make

```

```
[n]                                     root@user:/usr/src/linux-5.8.8
System is 8736 kB
CRC b5a11568
Kernel: arch/x86/boot/bzImage is ready  (#1)
MODPOST Module.symvers
CC [M]  drivers/thermal/intel/x86_pkg_temp_thermal.mod.o
LD [M]  drivers/thermal/intel/x86_pkg_temp_thermal.ko
CC [M]  fs/efivarfs/efivarfs.mod.o
LD [M]  fs/efivarfs/efivarfs.ko
CC [M]  net/ipv4/netfilter/iptable_nat.mod.o
LD [M]  net/ipv4/netfilter/iptable_nat.ko
CC [M]  net/ipv4/netfilter/nf_log_arp.mod.o
LD [M]  net/ipv4/netfilter/nf_log_arp.ko
CC [M]  net/ipv4/netfilter/nf_log_ipv4.mod.o
LD [M]  net/ipv4/netfilter/nf_log_ipv4.ko
CC [M]  net/ipv6/netfilter/nf_log_ipv6.mod.o
LD [M]  net/ipv6/netfilter/nf_log_ipv6.ko
CC [M]  net/netfilter/nf_log_common.mod.o
LD [M]  net/netfilter/nf_log_common.ko
CC [M]  net/netfilter/xt_LOG.mod.o
LD [M]  net/netfilter/xt_LOG.ko
CC [M]  net/netfilter/xt_MASQUERADE.mod.o
LD [M]  net/netfilter/xt_MASQUERADE.ko
CC [M]  net/netfilter/xt_addrtype.mod.o
LD [M]  net/netfilter/xt_addrtype.ko
CC [M]  net/netfilter/xt_mark.mod.o
LD [M]  net/netfilter/xt_mark.ko
CC [M]  net/netfilter/xt_nat.mod.o
LD [M]  net/netfilter/xt_nat.ko
root@user:/usr/src/linux-5.8.8# ]
```

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 grub.

\$ 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

```
[n]                                     root@user:/usr/src/linux-5.8.8
root@user:/usr/src/linux-5.8.8# make modules_install && make modules_install install
```

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
```

The screenshot shows a terminal window titled "root@user:/usr/src/linux-5.8.8". The terminal displays the following commands and output:

```
root@user:/usr/src/linux-5.8.8# ls
arch  COPYING  Documentation  include  Kbuild  lib      Makefile  README  security  usr
block CREDITS  drivers       init     Kconfig  LICENSES  mm      samples  sound    virt
certs  crypto   fs          ipc     kernel  MAINTAINERS  net     scripts  tools
root@user:/usr/src/linux-5.8.8# mkdir add_syscall
root@user:/usr/src/linux-5.8.8# ls
add_syscall  COPYING      drivers  ipc      lib      mm      scripts  usr
arch        CREDITS      fs       Kbuild  LICENSES  net      security  virt
block        crypto       include Kconfig  MAINTAINERS  README  sound
certs        Documentation init     kernel  Makefile  samples  tools
root@user:/usr/src/linux-5.8.8#
```

Step #2: Create the following files under add_syscall directory

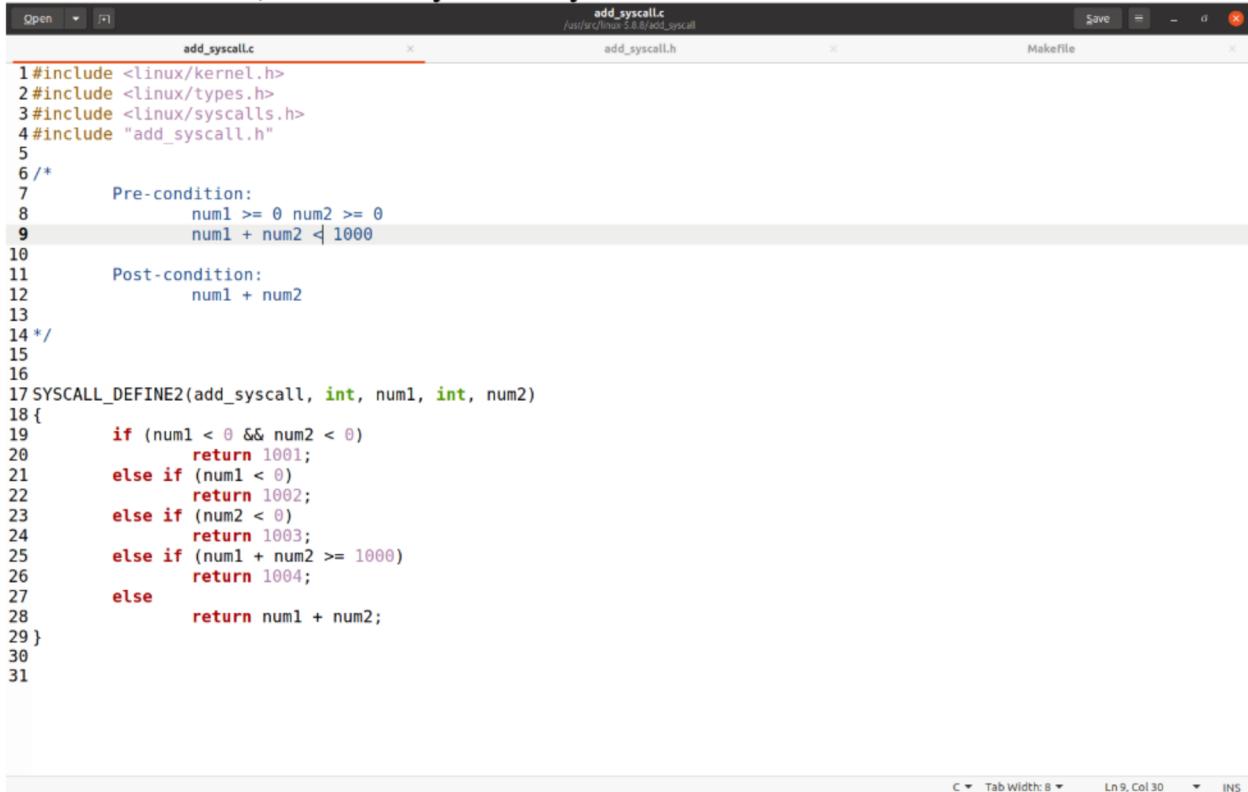
1. **add_syscall.c**
2. **add_syscall.h**
3. **Makefile**

Contents of **add_syscall.c**

SYSCALL_DEFINEn() 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.



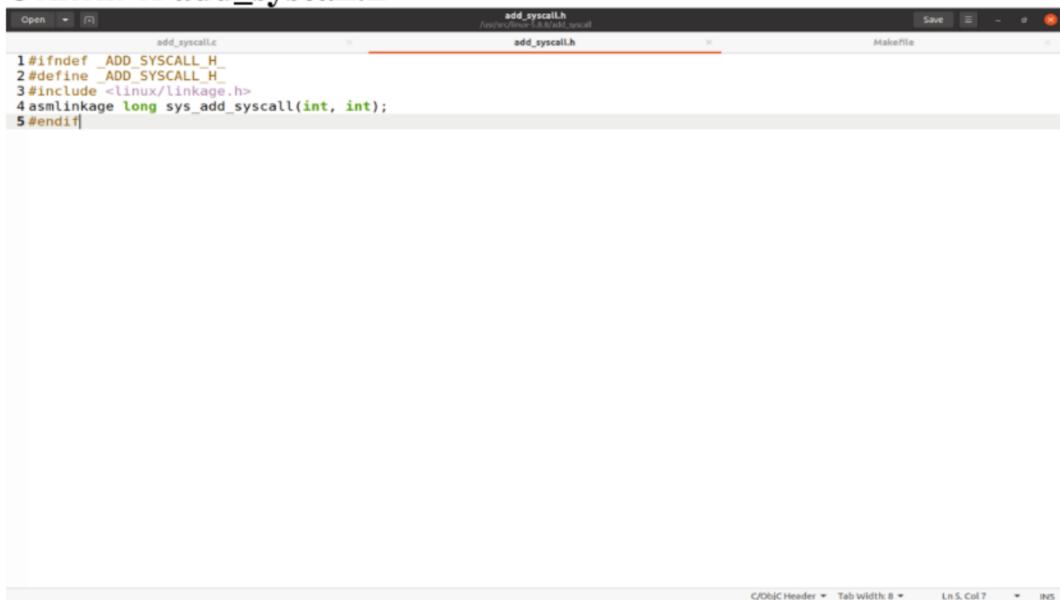
```

add_syscall.c
add_syscall.h
Makefile

1 #include <linux/kernel.h>
2 #include <linux/types.h>
3 #include <linux/syscalls.h>
4 #include "add_syscall.h"
5
6 /*
7     Pre-condition:
8         num1 >= 0 num2 >= 0
9         num1 + num2 < 1000
10
11    Post-condition:
12        num1 + num2
13
14 */
15
16
17 SYSCALL_DEFINE2(add_syscall, int, num1, int, num2)
18 {
19     if (num1 < 0 && num2 < 0)
20         return 1001;
21     else if (num1 < 0)
22         return 1002;
23     else if (num2 < 0)
24         return 1003;
25     else if (num1 + num2 >= 1000)
26         return 1004;
27     else
28         return num1 + num2;
29 }
30
31

```

Content of add_syscall.h



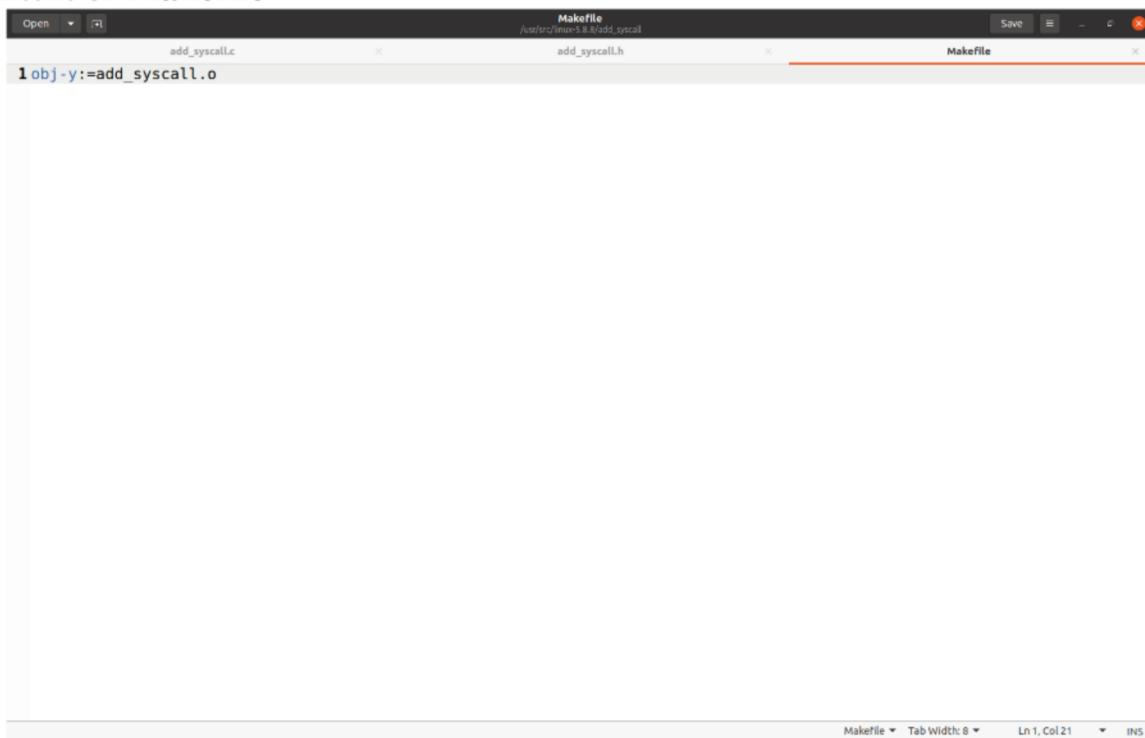
```

add_syscall.h
Makefile

1 #ifndef _ADD_SYSCALL_H_
2 #define _ADD_SYSCALL_H_
3 #include <linux/linkage.h>
4 asmlinkage long sys_add_syscall(int, int);
5#endif

```

Content of Makefile



```
1 obj-y:=add_syscall.o
```

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/**

```

1061 else
1062     SKIP_STACK_VALIDATION := 1
1063 export SKIP_STACK_VALIDATION
1064 endif
1065 endif
1066
1067 PHONY += prepare0
1068
1069 export MODORDER := $(extmod-prefix)modules.order
1070 export MODULES_NSDEPS := $(extmod-prefix)modules.nsdeps
1071
1072 ifeq ($(KBUILD_EXTMOD),)
1073 core-y      += kernel/ certs/ mm/ ipc/ security/ crypto/ block/ add_syscall/
1074
1075 vmlinux-dirs := $(patsubst %,%,$(filter %, \
1076             $(core-y) $(core-m) $(drivers-y) $(drivers-m) \
1077             $(libs-y) $(libs-m)))
1078
1079 vmlinux-alldirs      := $(sort $(vmlinux-dirs) Documentation \
1080             $(patsubst %,%,$(filter %, $(core-) \
1081                 $(drivers-) $(libs-))))
1082
1083 subdir-modorder := $(addsuffix modules.order,$(filter %, \
1084             $(core-y) $(core-m) $(libs-y) $(libs-m) \
1085             $(drivers-y) $(drivers-m)))
1086
1087 build-dirs      := $(VMLINUX_DIRS)
1088 clean-dirs      := $(VMLINUX_ALLDIRS)
1089
1090 # Externally visible symbols (used by link-vmlinux.sh)
1091 KBUILD_VMLINUX_OBJS := $(head-y) $(patsubst %,%/built-in.a, $(core-y))
1092 KBUILD_VMLINUX_OBJS += $(addsuffix built-in.a, $(filter %, $(libs-y)))
1093 ifdef CONFIG_MODULES
1094 KBUILD_VMLINUX_OBJS += $(patsubst %, %/lib.a, $(filter %, $(libs-y)))
1095 KBUILD_VMLINUX_LIBS := $(filter-out %, $(libs-y))
1096 else
1097 KBUILD_VMLINUX_LIBS := $(patsubst %/ %/lib.a %/libc-v1)

```

Makefile Tab Width: 8 Ln 1073, Col 85 INS

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.

Open Save

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

```

333 322    64    execveat      sys_execveat
334 323    common userfaultfd sys_userfaultfd
335 324    common membarrier   sys_membarrier
336 325    common mlock2       sys_mlock2
337 326    common copy_file_range sys_copy_file_range
338 327    64    preadv2       sys_preadv2
339 328    64    pwritev2      sys_pwritev2
340 329    common pkey_mprotect sys_pkey_mprotect
341 330    common pkey_alloc    sys_pkey_alloc
342 331    common pkey_free     sys_pkey_free
343 332    common statx        sys_statx
344 333    common io_pgetevents sys_io_pgetevents
345 334    common rseq         sys_rseq
346 # don't use numbers 387 through 423, add new calls after the last
347 # 'common' entry
348 424    common pidfd_send_signal sys_pidfd_send_signal
349 425    common io_uring_setup   sys_io_uring_setup
350 426    common io_uring_enter   sys_io_uring_enter
351 427    common io_uring_register sys_io_uring_register
352 428    common open_tree       sys_open_tree
353 429    common move_mount     sys_move_mount
354 430    common fsopen         sys_fsopen
355 431    common fsconfig        sys_fsconfig
356 432    common fsmount        sys_fsmount
357 433    common fspick         sys_fspick
358 434    common pidfd_open     sys_pidfd_open
359 435    common clone3          sys_clone3
360 437    common openat2        sys_openat2
361 438    common pidfd_getfd   sys_pidfd_getfd
362 439    common faccessat2    sys_faccessat2
363 440    common add_syscall   sys_add_syscall
364
365 #
366 # x32-specific system call numbers start at 512 to avoid cache impact
367 # for native 64-bit operation. The __x32_compat_sys stubs are created
368 # on-the-fly for compat_sys_*() compatibility system calls if X86_X32
369 # is defined
Saving file "/usr/src/linux-5.8.8/arch/x86/entry/syscalls/syscall_64.tbl"...
Plain Text Tab Width: 8 Ln 363, Col 56 INS

```

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:

Open Save

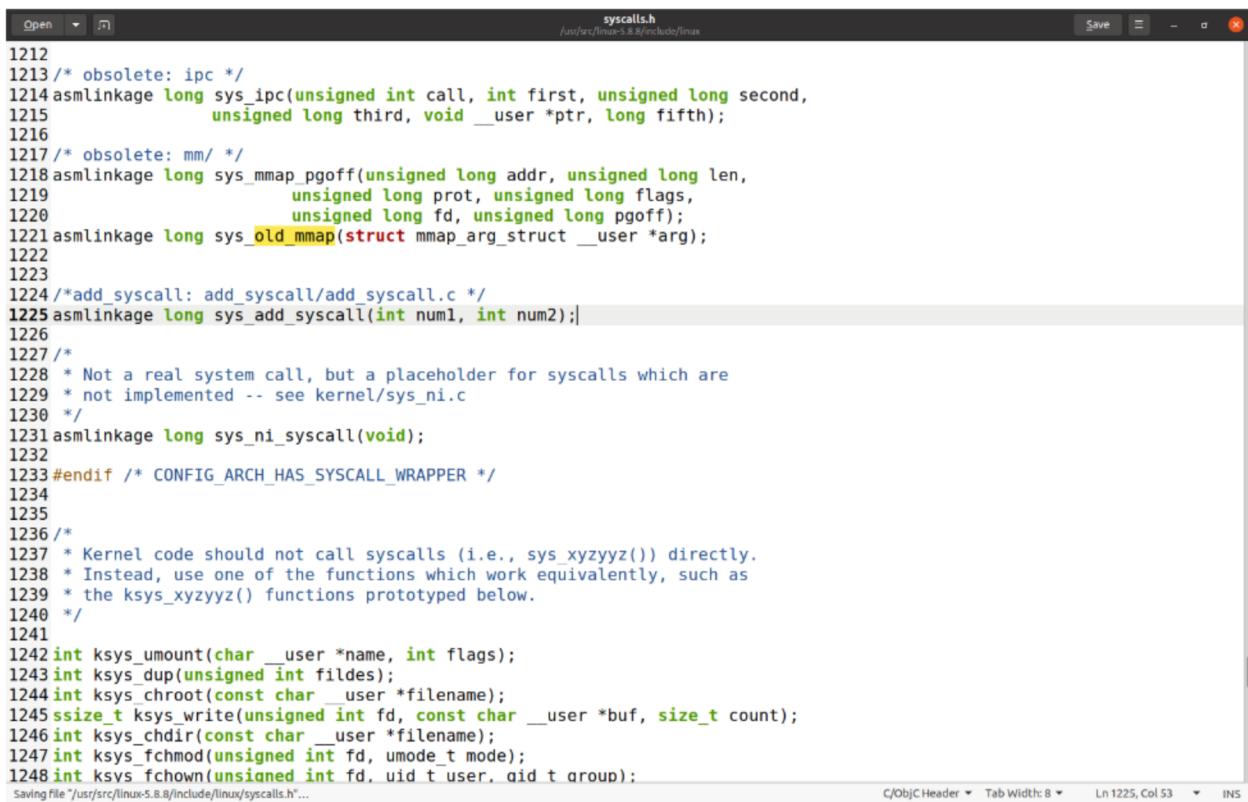
*syscalls.h
/usr/src/linux-5.8.8/include/asm-generic

```

1 /* SPDX-License-Identifier: GPL-2.0 */
2 #ifndef __ASM_GENERIC_SYSCALLS_H
3 #define __ASM_GENERIC_SYSCALLS_H
4
5 #include <linux/compiler.h>
6 #include <linux/linkage.h>
7
8 /*
9  * Calling conventions for these system calls can differ, so
10 * it's possible to override them.
11 */
12
13 #ifndef sys_mmap2
14 asmlinkage long sys_mmap2(unsigned long addr, unsigned long len,
15                           unsigned long prot, unsigned long flags,
16                           unsigned long fd, unsigned long pgoff);
17#endif
18
19 #ifndef sys_mmap
20 asmlinkage long sys_mmap(unsigned long addr, unsigned long len,
21                         unsigned long prot, unsigned long flags,
22                         unsigned long fd, off_t pgoff);
23#endif
24
25 #ifndef sys_rt_sigreturn
26 asmlinkage long sys_rt_sigreturn(struct pt_regs *regs);
27#endif
28
29 #ifndef sys_add_syscall
30 asmlinkage long sys_add_syscall(int num1, int num2);
31#endif
32
33#endif /* __ASM_GENERIC_SYSCALLS_H */

```

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



```
1212 /* obsolete: ipc */
1213 long sys_ipc(unsigned int call, int first, unsigned long second,
1214                 unsigned long third, void __user *ptr, long fifth);
1215
1216 /* obsolete: mm/ */
1217 long sys_mmap_pgoff(unsigned long addr, unsigned long len,
1218                      unsigned long prot, unsigned long flags,
1219                      unsigned long fd, unsigned long pgoff);
1220 asmlinkage long sys_old_mmap(struct mmap_arg_struct __user *arg);
1221
1222 /*add_syscall: add_syscall/add_syscall.c */
1223 asmlinkage long sys_add_syscall(int num1, int num2);
1224
1225 /*
1226  * Not a real system call, but a placeholder for syscalls which are
1227  * not implemented -- see kernel/sys_ni.c
1228  */
1229 asmlinkage long sys_ni_syscall(void);
1230
1231 #endif /* CONFIG_ARCH_HAS_SYSCALL_WRAPPER */
1232
1233
1234
1235
1236 /*
1237  * Kernel code should not call syscalls (i.e., sys_xyzzyz()) directly.
1238  * Instead, use one of the functions which work equivalently, such as
1239  * the ksys_xyzzyz() functions prototyped below.
1240  */
1241
1242 int ksys_umount(char __user *name, int flags);
1243 int ksys_dup(unsigned int fildes);
1244 int ksys_chroot(const char __user *filename);
1245 ssize_t ksys_write(unsigned int fd, const char __user *buf, size_t count);
1246 int ksys_chdir(const char __user *filename);
1247 int ksys_fchmod(unsigned int fd, umode_t mode);
1248 int ksys_fchown(unsigned int fd, uid_t user, gid_t group);
Saving file "/usr/src/linux-5.8.8/include/linux/syscalls.h..."
```

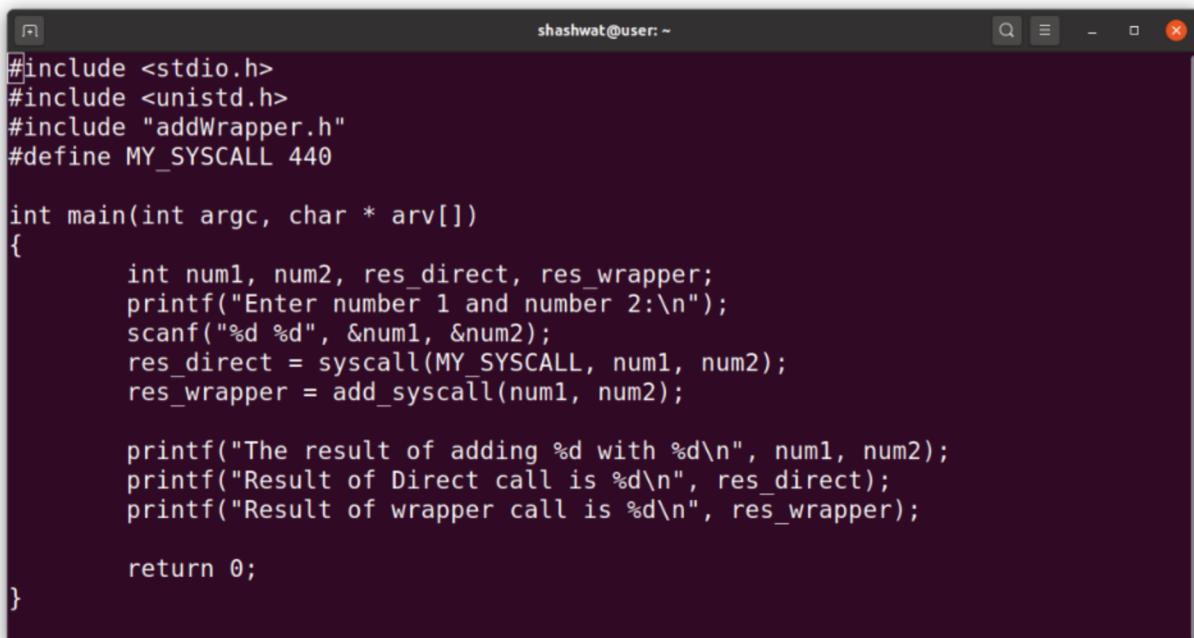
Recompile the Kernel [Follow section#2]to get all the changes reflected. Reboot the system and boot into this kernel from the grub <Select advanced ubuntu tab followed by the New kernel>

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:



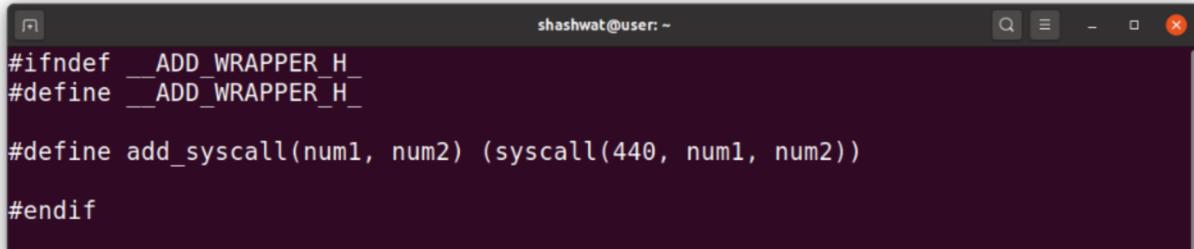
```
#include <stdio.h>
#include <unistd.h>
#include "addWrapper.h"
#define MY_SYSCALL 440

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



```
#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:

```
shashwat@user:~$ gcc -o addNum add2Num.c
shashwat@user:~$ ./addNum
Enter number 1 and number 2:
10 20
The result of adding 10 with 20
Result of Direct call is 30
Result of wrapper call is 30
shashwat@user:~$ ./addNum
Enter number 1 and number 2:
-2 -5
The result of adding -2 with -5
Result of Direct call is 1001
Result of wrapper call is 1001
shashwat@user:~$ ./addNum
Enter number 1 and number 2:
2 -5
The result of adding 2 with -5
Result of Direct call is 1003
Result of wrapper call is 1003
shashwat@user:~$ ./addNum
Enter number 1 and number 2:
-2 5
The result of adding -2 with 5
Result of Direct call is 1002
Result of wrapper call is 1002
shashwat@user:~$ ./addNum
Enter number 1 and number 2:
500 600
The result of adding 500 with 600
Result of Direct call is 1004
Result of wrapper call is 1004
shashwat@user:~$
```

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.