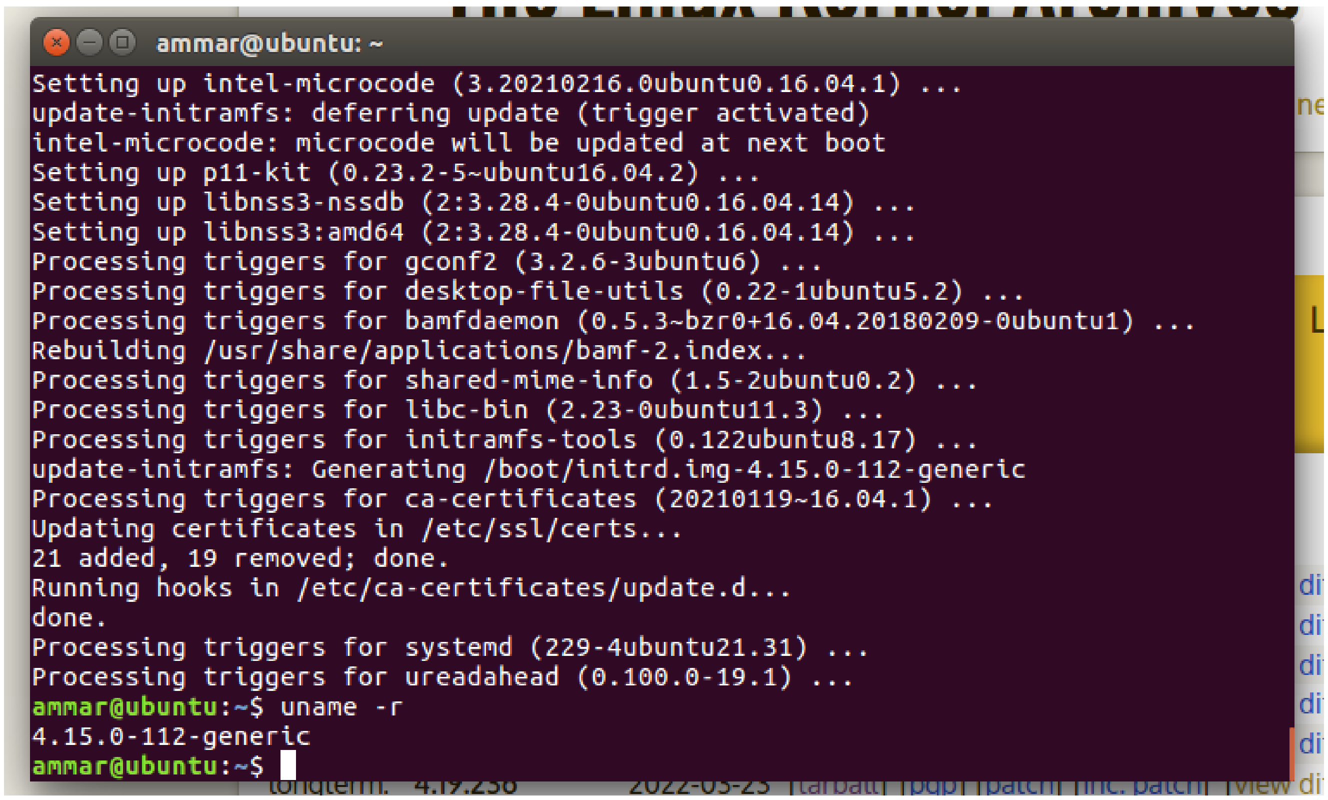
The Ubuntu version on which I am configuring the kernel is 16.04

STEP 01:

Open the terminal by pressing (Ctrl + Alt + T) or by searching it in your linux and type the following commands:

1) sudo apt-get install gcc
2) sudo apt-get install libncurses5-dev
3) sudo apt-get install bison
4) sudo apt-get install flex
5) sudo apt install make
6) sudo apt-get install libssl-dev
7) sudo apt-get install libelf-dev
8) sudo add-apt-repository "deb http://archive.ubuntu.com/ubuntu $(lsb_release -sc) main universe"
9) sudo apt-get update
10) sudo apt-get upgrade

The current version of my kernel is 4.15.0-112. Therefore I am upgrading it to 4.19

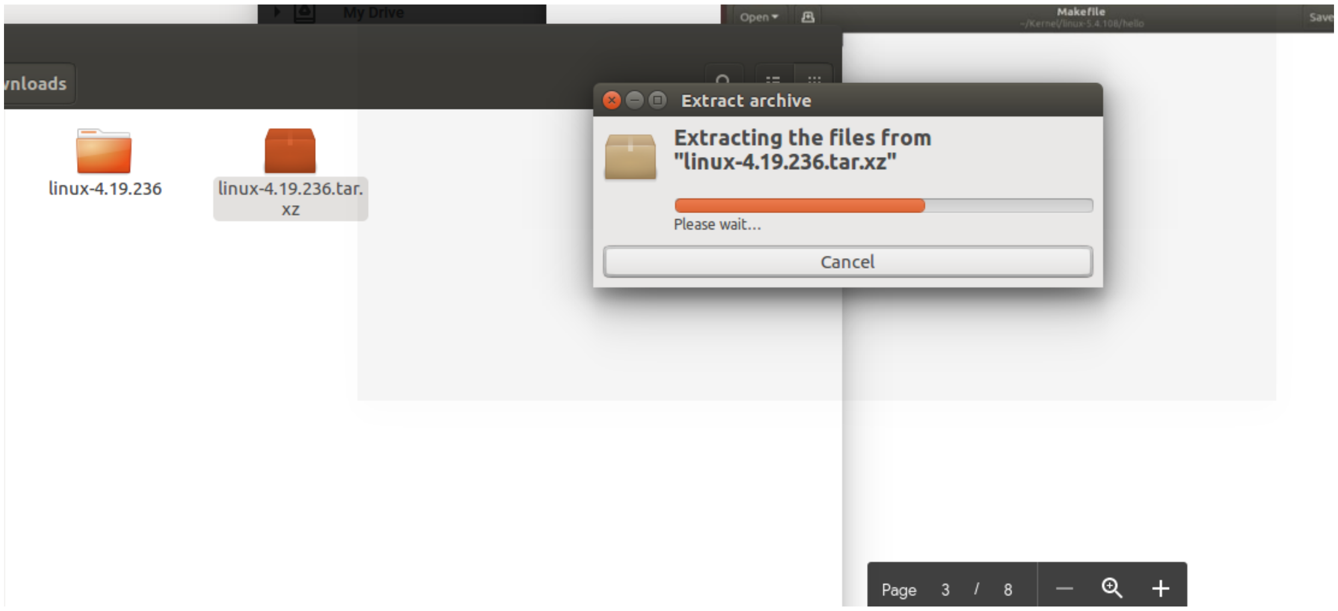


```
Setting up intel-microcode (3.20210216.0ubuntu0.16.04.1) ...
update-initramfs: deferring update (trigger activated)
intel-microcode: microcode will be updated at next boot
Setting up p11-kit (0.23.2-5~ubuntu16.04.2) ...
Setting up libnss3-nssdb (2:3.28.4-0ubuntu0.16.04.14) ...
Setting up libnss3:amd64 (2:3.28.4-0ubuntu0.16.04.14) ...
Processing triggers for gconf2 (3.2.6-3ubuntu6) ...
Processing triggers for desktop-file-utils (0.22-1ubuntu5.2) ...
Processing triggers for bamfdaemon (0.5.3~bzr0+16.04.20180209-0ubuntu1) ...
Rebuilding /usr/share/applications/bamf-2.index...
Processing triggers for shared-mime-info (1.5-2ubuntu0.2) ...
Processing triggers for libc-bin (2.23-0ubuntu11.3) ...
Processing triggers for initramfs-tools (0.122ubuntu8.17) ...
update-initramfs: Generating /boot/initrd.img-4.15.0-112-generic
Processing triggers for ca-certificates (20210119~16.04.1) ...
Updating certificates in /etc/ssl/certs...
21 added, 19 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
Processing triggers for systemd (229-4ubuntu21.31) ...
Processing triggers for ureadahead (0.100.0-19.1) ...
ammar@ubuntu:~$ uname -r
4.15.0-112-generic
ammar@ubuntu:~$
```

STEP 02:

Downloading the Kernel version (4.19) through Tarball link.

Extracting the tarr file

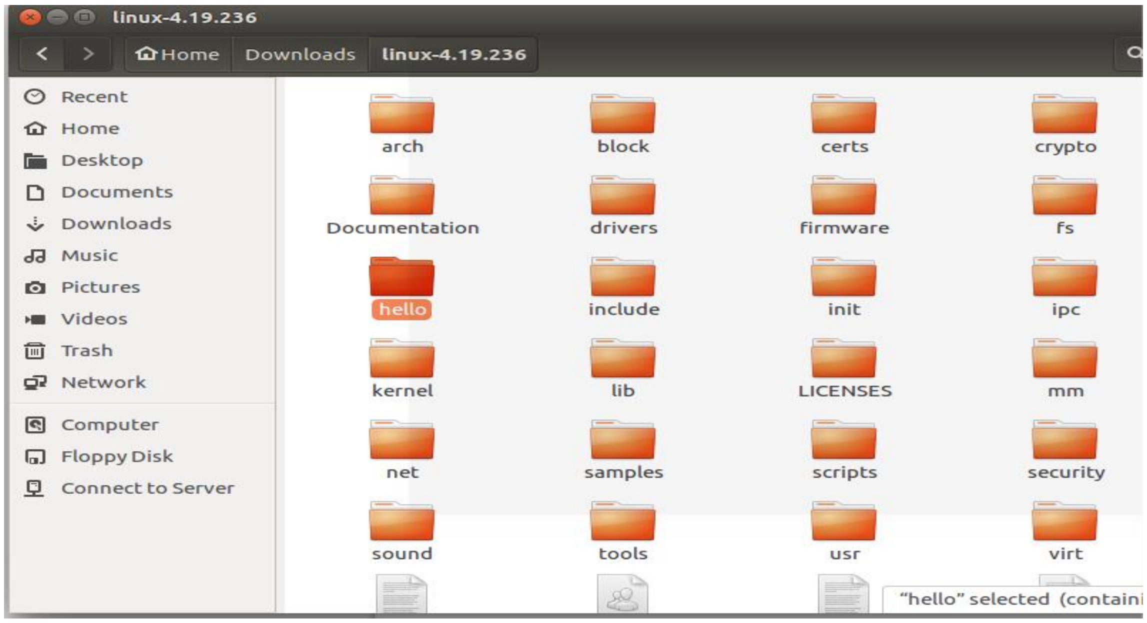


Extracting the files from "linux-4.19.236.tar.xz"

Please wait...

STEP 03:

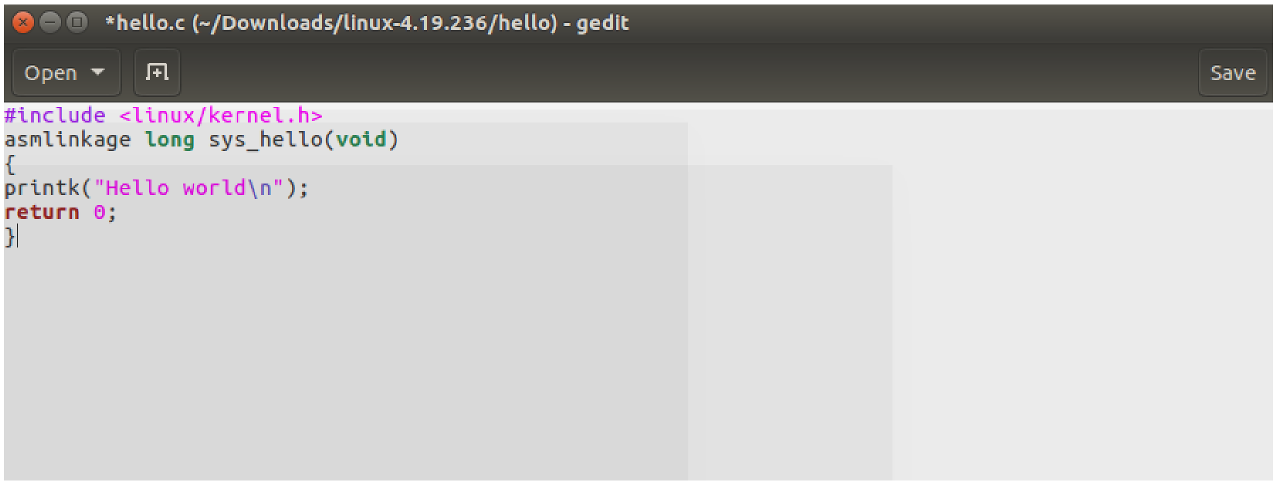
Making a new folder named “hello” in the extracted folder.



STEP 04:

Making a file named “hello.c” in the hello directory through “gedit hello.c” command
Write down the following code in that file:

```
#include <linux/kernel.h>
asmlinkage long sys_hello(void)
{
        printk(“Hello world\n”);
        return 0;
}
```

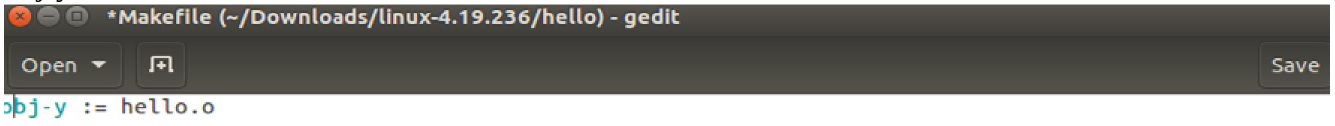


```
#include <linux/kernel.h>
asmlinkage long sys_hello(void)
{
printk("Hello world\n");
return 0;
}
```

STEP 05:

Create a MakeFile for our hello.c file so that object file is created everytime we compile our Kernel and write down the following code:

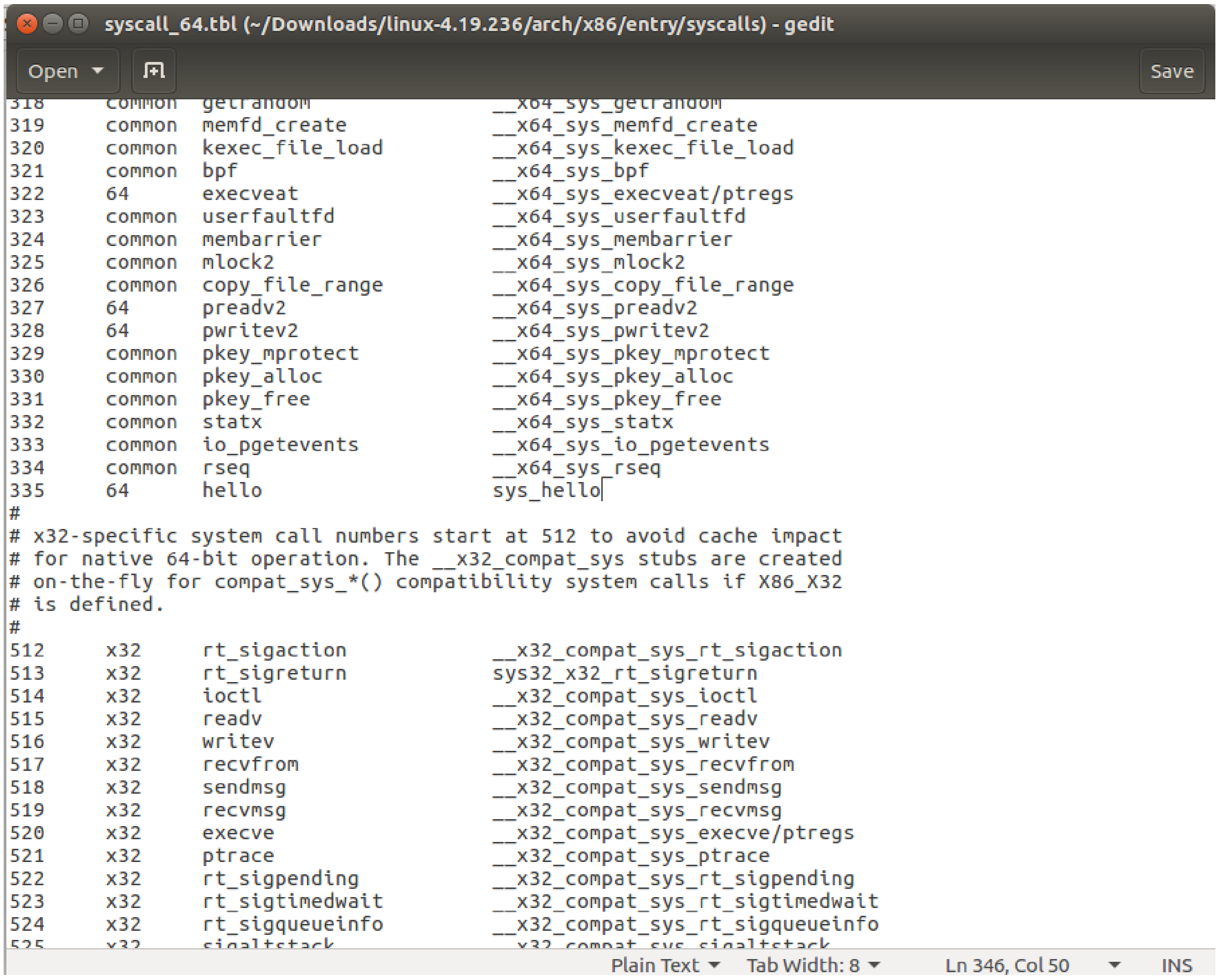
obj-y := hello.o



```
obj-y := hello.o
```

STEP 06:

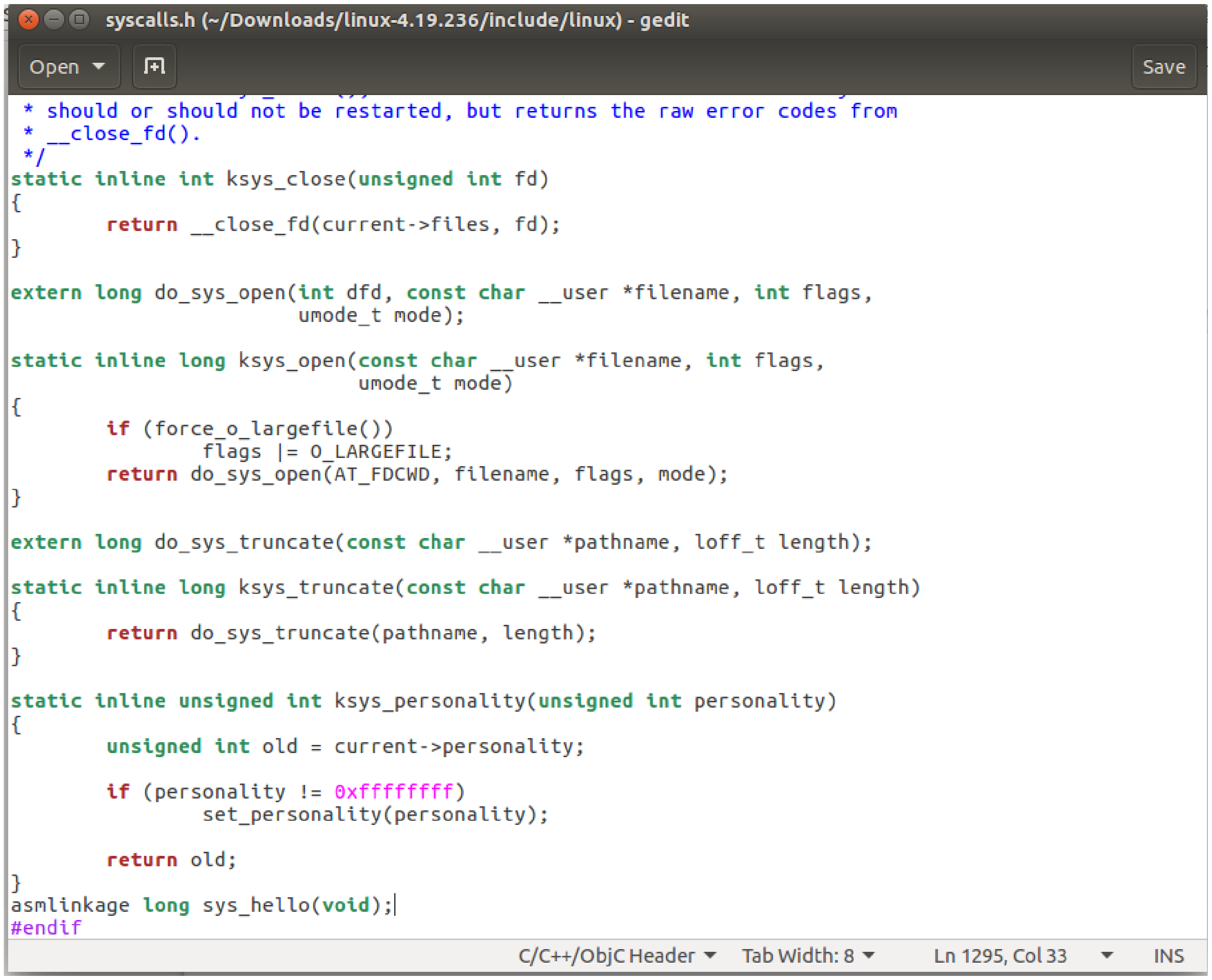
Now we need to add our system call in our syscall_64.tbl. The table is located inside the kernel folder in arch/x86/entry/syscalls. First we will move to this directory and open the file by using command “gedit syscall_64.tbl” and add our system call in the table as shown in picture below.


syscall_64.tbl (~/Downloads/linux-4.19.236/arch/x86/entry/syscalls) - gedit
Open
Save
Plain Text
Tab Width: 8
Ln 346, Col 50
INS

```
318	common	getrandom		__x64_sys_getrandom
319	common	memfd_create		__x64_sys_memfd_create
320	common	kexec_file_load		__x64_sys_kexec_file_load
321	common	bpf			__x64_sys_bpf
322	64	execveat		__x64_sys_execveat/ptregs
323	common	userfaultfd		__x64_sys_userfaultfd
324	common	membarrier		__x64_sys_membarrier
325	common	mlock2			__x64_sys_mlock2
326	common	copy_file_range		__x64_sys_copy_file_range
327	64	preadv2			__x64_sys_preadv2
328	64	pwritev2		__x64_sys_pwritev2
329	common	pkey_mprotect		__x64_sys_pkey_mprotect
330	common	pkey_alloc		__x64_sys_pkey_alloc
331	common	pkey_free		__x64_sys_pkey_free
332	common	statx			__x64_sys_statx
333	common	io_pgetevents		__x64_sys_io_pgetevents
334	common	rseq			__x64_sys_rseq
335	64	hello			sys_hello
#
# x32-specific system call numbers start at 512 to avoid cache impact
# for native 64-bit operation. The __x32_compat_sys stubs are created
# on-the-fly for compat_sys_*() compatibility system calls if X86_X32
# is defined.
#
512	x32	rt_sigaction		__x32_compat_sys_rt_sigaction
513	x32	rt_sigreturn		sys32_x32_rt_sigreturn
514	x32	ioctl			__x32_compat_sys_ioctl
515	x32	readv			__x32_compat_sys_readv
516	x32	writev			__x32_compat_sys_writev
517	x32	recvfrom		__x32_compat_sys_recvfrom
518	x32	sendmsg			__x32_compat_sys_sendmsg
519	x32	recvmsg			__x32_compat_sys_recvmsg
520	x32	execve			__x32_compat_sys_execve/ptregs
521	x32	ptrace			__x32_compat_sys_ptrace
522	x32	rt_sigpending		__x32_compat_sys_rt_sigpending
523	x32	rt_sigtimedwait		__x32_compat_sys_rt_sigtimedwait
524	x32	rt_sigqueueinfo		__x32_compat_sys_rt_sigqueueinfo
525	x32	sigaltstack		__x32_compat_sys_sigaltstack
```

STEP 07:

Now we will add the prototype of of our system call in the system header's file which is located in the kernel folder in the following path "/include/linux". In this folder we will open the file syscalls.h using "gedit syscalls.h" and add our function prototype at the end of the file as shown below.



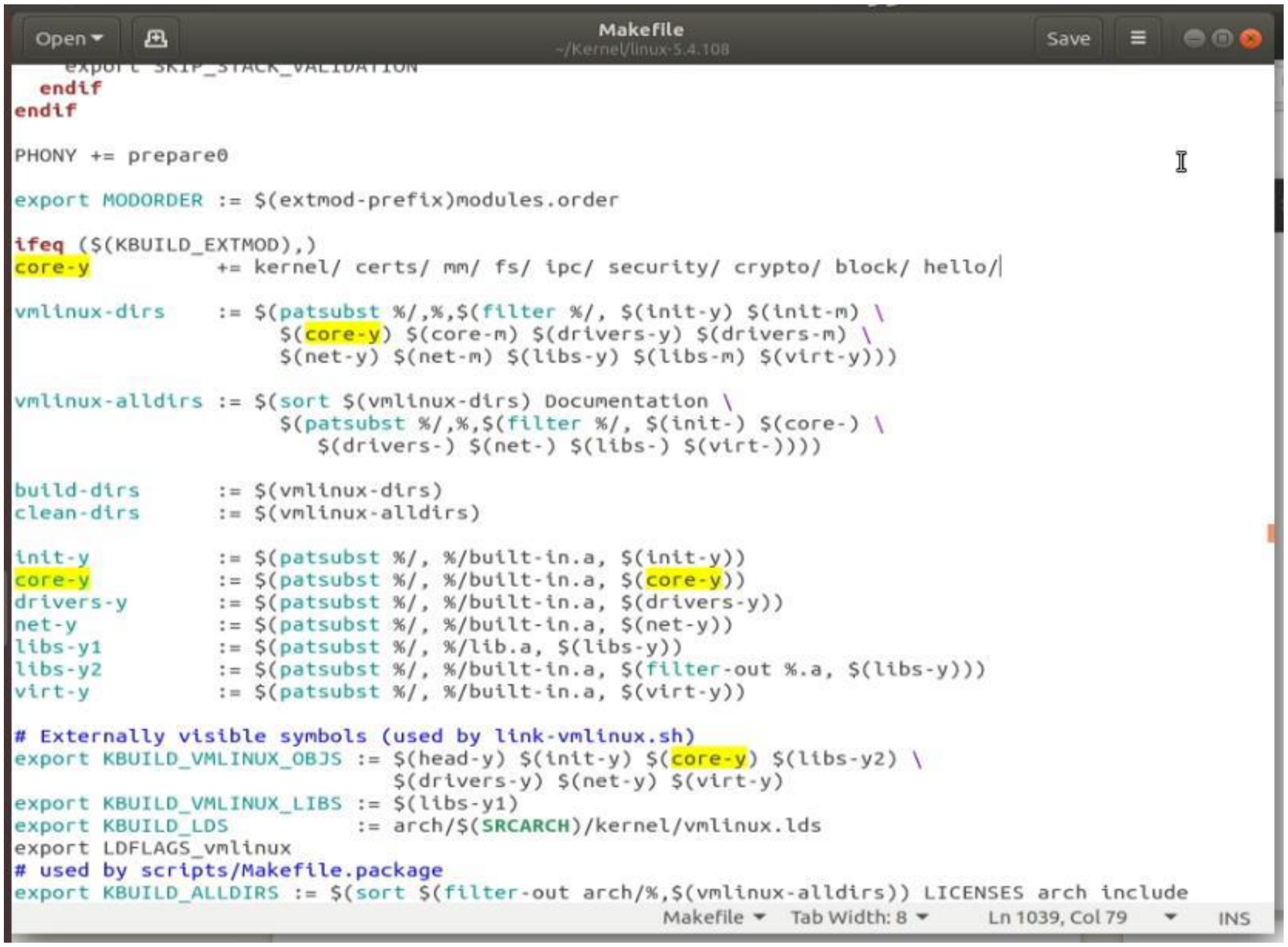
syscalls.h (~/Downloads/linux-4.19.236/include/linux) - gedit

```
 * should or should not be restarted, but returns the raw error codes from
 * __close_fd().
 */
static inline int ksys_close(unsigned int fd)
{
	return __close_fd(current->files, fd);
}

extern long do_sys_open(int dfd, const char __user *filename, int flags,
			umode_t mode);

static inline long ksys_open(const char __user *filename, int flags,
			     umode_t mode)
{
	if (force_o_largefile())
		flags |= O_LARGEFILE;
	return do_sys_open(AT_FDCWD, filename, flags, mode);
}

extern long do_sys_truncate(const char __user *pathname, loff_t length);

static inline long ksys_truncate(const char __user *pathname, loff_t length)
{
	return do_sys_truncate(pathname, length);
}

static inline unsigned int ksys_personality(unsigned int personality)
{
	unsigned int old = current->personality;

	if (personality != 0xffffffff)
		set_personality(personality);

	return old;
}
asmlinkage long sys_hello(void);
#endif
```

C/C++/ObjC Header ▾ Tab Width: 8 ▾ Ln 1295, Col 33 ▾ INS

STEP 08:

Then we will change the kernel version and also will add our roll number in the extraversion section. (Note: We can only add numbers in extraversion section starting with a '-' sign, in my case it was -200177). After that we will press Ctrl+f and search "core-y" and in it's second instance, we will add the name of the (.c) file we created after "block/" in the same manner as shown below.

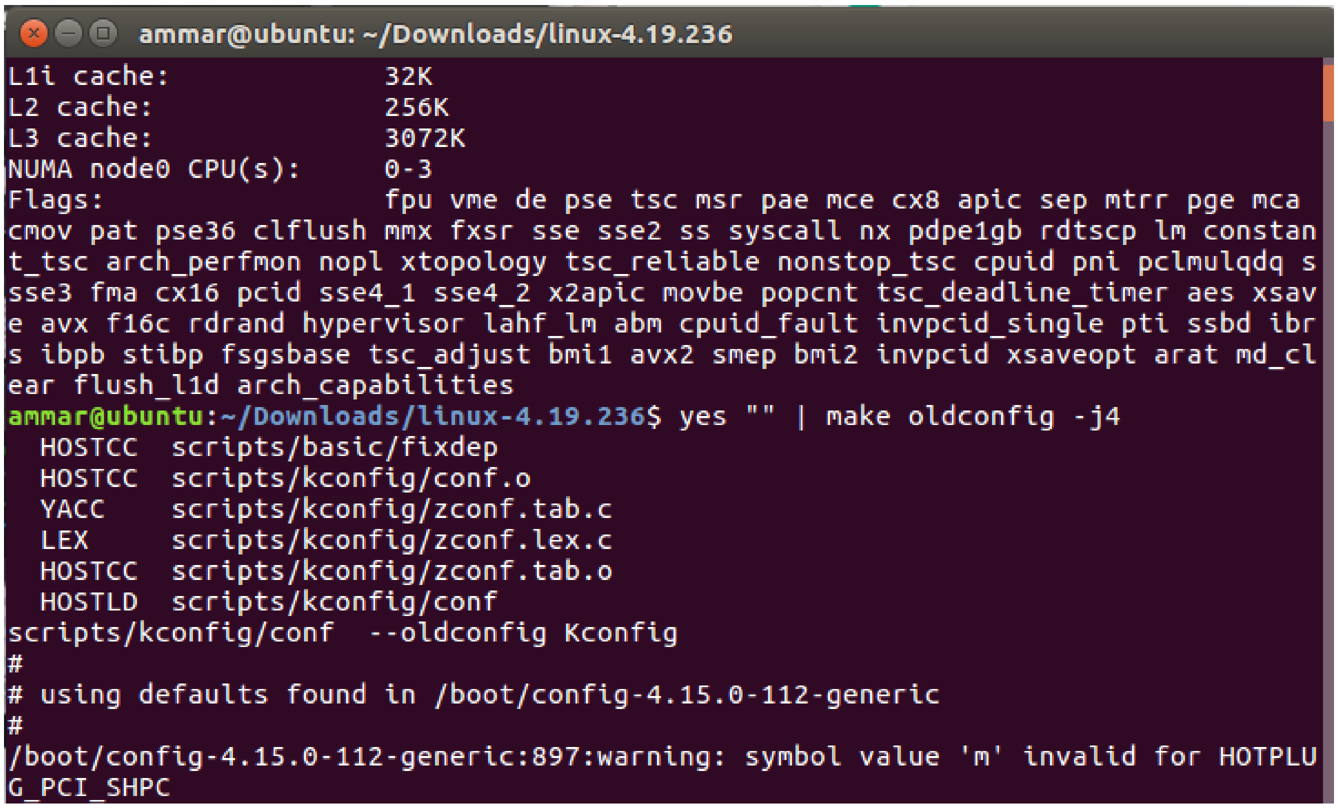


```
  endif
endif

PHONY += prepare0

export MODORDER := $(extmod-prefix)modules.order

ifeq ($(KBUILD_EXTMOD),)
core-y          += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ hello/

vmlinux-dirs    := $(patsubst %/,%,$(filter %/, $(init-y) $(init-m) \
                     $(core-y) $(core-m) $(drivers-y) $(drivers-m) \
                     $(net-y) $(net-m) $(libs-y) $(libs-m) $(virt-y)))

vmlinux-alldirs := $(sort $(vmlinux-dirs) Documentation \
                     $(patsubst %/,%,$(filter %/, $(init-) $(core-) \
                        $(drivers-) $(net-) $(libs-) $(virt-))))

build-dirs      := $(vmlinux-dirs)
clean-dirs      := $(vmlinux-alldirs)

init-y          := $(patsubst %/, %/built-in.a, $(init-y))
core-y          := $(patsubst %/, %/built-in.a, $(core-y))
drivers-y       := $(patsubst %/, %/built-in.a, $(drivers-y))
net-y           := $(patsubst %/, %/built-in.a, $(net-y))
libs-y1         := $(patsubst %/, %/lib.a, $(libs-y))
libs-y2         := $(patsubst %/, %/built-in.a, $(filter-out %.a, $(libs-y)))
virt-y          := $(patsubst %/, %/built-in.a, $(virt-y))

# Externally visible symbols (used by link-vmlinux.sh)
export KBUILD_VMLINUX_OBJS := $(head-y) $(init-y) $(core-y) $(libs-y2) \
                              $(drivers-y) $(net-y) $(virt-y)
export KBUILD_VMLINUX_LIBS := $(libs-y1)
export KBUILD_LDS          := arch/$(SRCARCH)/kernel/vmlinux.lds
export LDFLAGS_vmlinux
# used by scripts/Makefile.package
export KBUILD_ALLDIRS := $(sort $(filter-out arch/%,$(vmlinux-alldirs)) LICENSES arch include
```

STEP 09:

After this we will create a new config file for our kernel. To do this we will copy the oldconfig and use it for our new kernel, to perform this task we will follow the following steps:

- Search config by typing "ls /boot | grep config" and then we copy the config that is shown to us and type "cp /boot/config-4.15.0-112 -generic *our linux kernel directory*".
- Then we will create old config by typing "yes" | make oldconfig -j4"



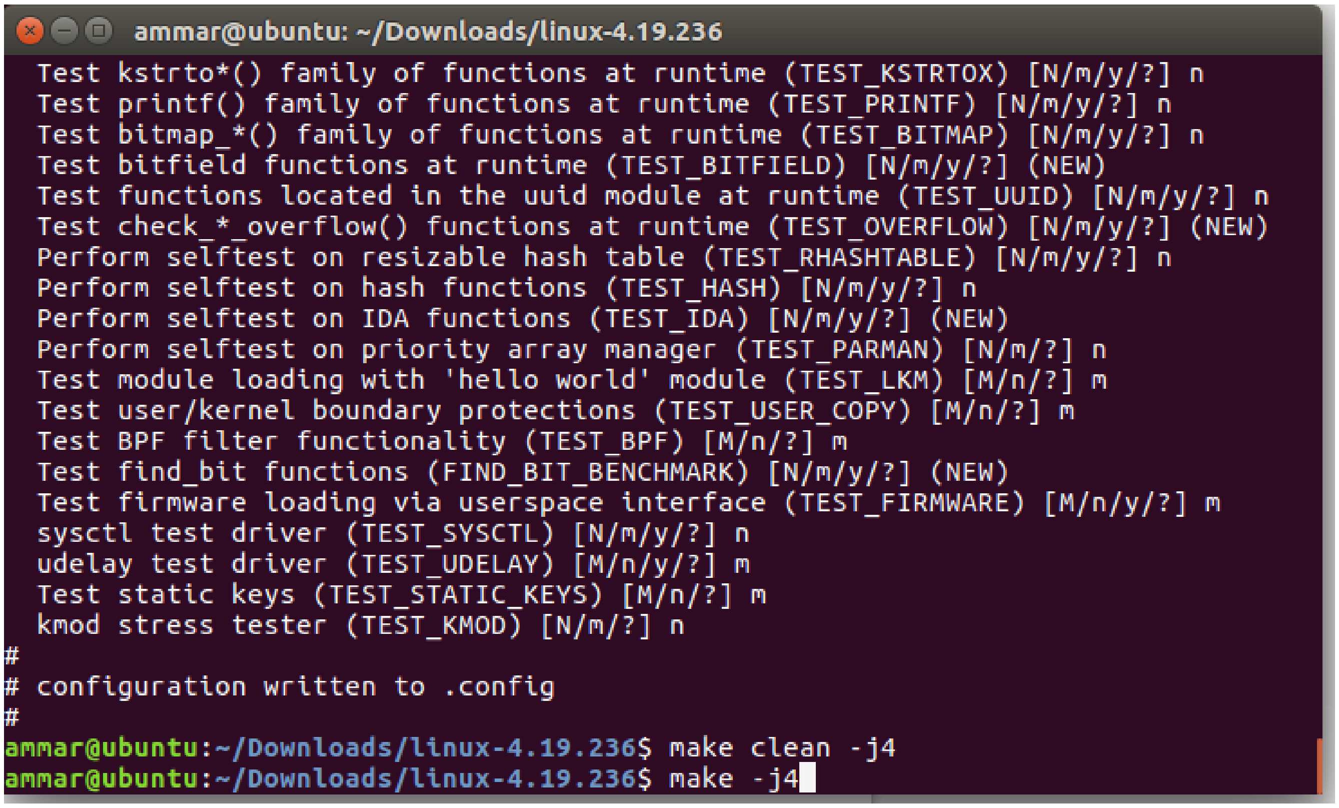
```
ammar@ubuntu: ~/Downloads/linux-4.19.236
L1i cache:              32K
L2 cache:               256K
L3 cache:               3072K
NUMA node0 CPU(s):      0-3
Flags:                  fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pat pse36 clflush mmx fxsr sse sse2 ss syscall nx pdpe1gb rdtscp lm constan
t_tsc arch_perfmon nopl xtopology tsc_reliable nonstop_tsc cpuid pni pclmulqdq s
sse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsav
e avx f16c rdrand hypervisor lahf_lm abm cpuid_fault invpcid_single pti ssbd ibr
s ibpb stibp fsgsbase tsc_adjust bmi1 avx2 smep bmi2 invpcid xsaveopt arat md_cl
ear flush_l1d arch_capabilities
ammar@ubuntu:~/Downloads/linux-4.19.236$ yes "" | make oldconfig -j4
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  YACC    scripts/kconfig/zconf.tab.c
  LEX     scripts/kconfig/zconf.lex.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
scripts/kconfig/conf  --oldconfig Kconfig
#
# using defaults found in /boot/config-4.15.0-112-generic
#
/boot/config-4.15.0-112-generic:897:warning: symbol value 'm' invalid for HOTPLU
G_PCI_SHPC
```

STEP 10:

Now we will clean and then will compile our kernel,

For cleaning we will use “make clean -j4” command.

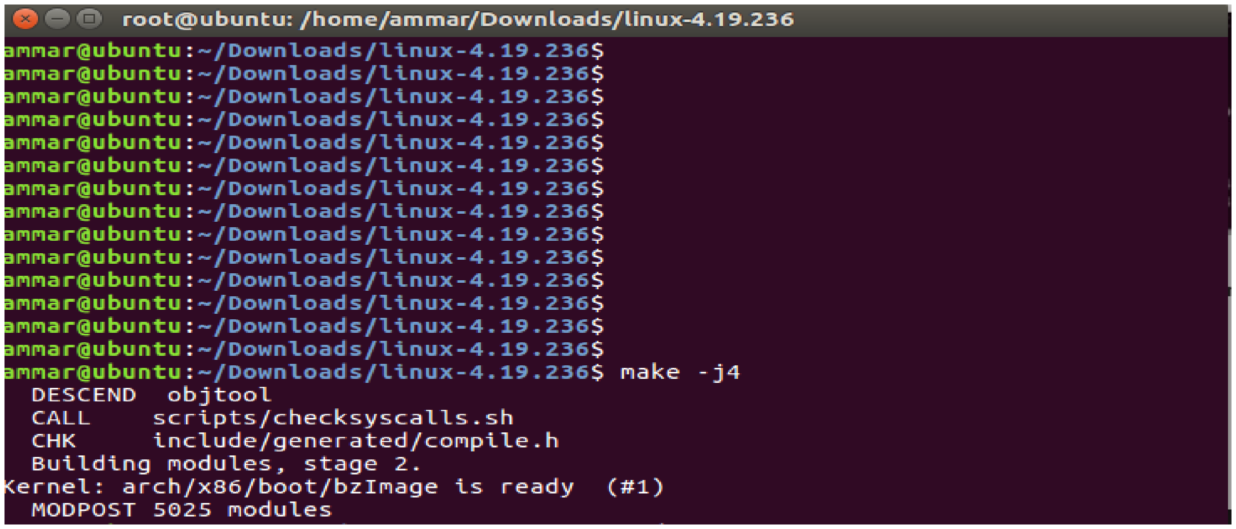
For making/compiling our kernel we will use “make -j4” (Here j4 is the number of CPU’s we are giving to our kernel. I had 4 of them so I gave it all 4 so that it may compile faster). After this we will have to wait until our kernel gets compiled (put the laptop on charge and go to sleep)



```
  Test kstrto*() family of functions at runtime (TEST_KSTRTOX) [N/m/y/?] n
  Test printf() family of functions at runtime (TEST_PRINTF) [N/m/y/?] n
  Test bitmap_*() family of functions at runtime (TEST_BITMAP) [N/m/y/?] n
  Test bitfield functions at runtime (TEST_BITFIELD) [N/m/y/?] (NEW)
  Test functions located in the uuid module at runtime (TEST_UUID) [N/m/y/?] n
  Test check_*_overflow() functions at runtime (TEST_OVERFLOW) [N/m/y/?] (NEW)
  Perform selftest on resizable hash table (TEST_RHASHTABLE) [N/m/y/?] n
  Perform selftest on hash functions (TEST_HASH) [N/m/y/?] n
  Perform selftest on IDA functions (TEST_IDA) [N/m/y/?] (NEW)
  Perform selftest on priority array manager (TEST_PARMAN) [N/m/?] n
  Test module loading with 'hello world' module (TEST_LKM) [M/n/?] m
  Test user/kernel boundary protections (TEST_USER_COPY) [M/n/?] m
  Test BPF filter functionality (TEST_BPF) [M/n/?] m
  Test find_bit functions (FIND_BIT_BENCHMARK) [N/m/y/?] (NEW)
  Test firmware loading via userspace interface (TEST_FIRMWARE) [M/n/y/?] m
  sysctl test driver (TEST_SYSCTL) [N/m/y/?] n
  udelay test driver (TEST_UDELAY) [M/n/y/?] m
  Test static keys (TEST_STATIC_KEYS) [M/n/?] m
  kmod stress tester (TEST_KMOD) [N/m/?] n
#
# configuration written to .config
#
ammar@ubuntu:~/Downloads/linux-4.19.236$ make clean -j4
ammar@ubuntu:~/Downloads/linux-4.19.236$ make -j4
```

STEP 11:

After compiling we can check if the compilation was successful or not by typing "make -j4" again and see if the below output is printed on kernel

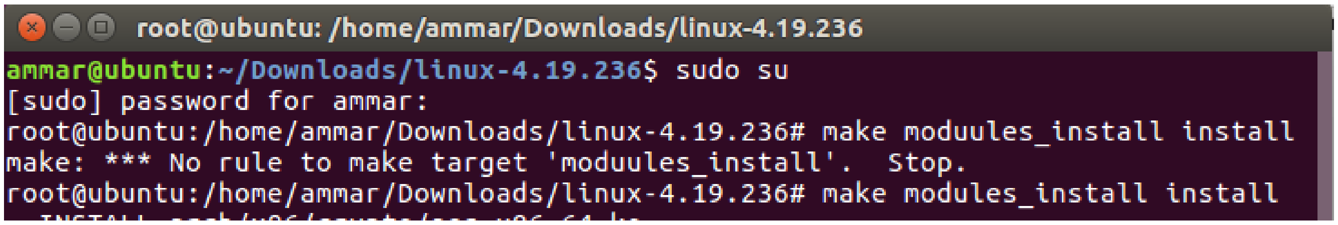


```
root@ubuntu: /home/ammar/Downloads/linux-4.19.236
ammar@ubuntu:~/Downloads/linux-4.19.236$
ammar@ubuntu:~/Downloads/linux-4.19.236$
ammar@ubuntu:~/Downloads/linux-4.19.236$
ammar@ubuntu:~/Downloads/linux-4.19.236$
ammar@ubuntu:~/Downloads/linux-4.19.236$
ammar@ubuntu:~/Downloads/linux-4.19.236$
ammar@ubuntu:~/Downloads/linux-4.19.236$
ammar@ubuntu:~/Downloads/linux-4.19.236$
ammar@ubuntu:~/Downloads/linux-4.19.236$
ammar@ubuntu:~/Downloads/linux-4.19.236$
ammar@ubuntu:~/Downloads/linux-4.19.236$
ammar@ubuntu:~/Downloads/linux-4.19.236$
ammar@ubuntu:~/Downloads/linux-4.19.236$
ammar@ubuntu:~/Downloads/linux-4.19.236$
ammar@ubuntu:~/Downloads/linux-4.19.236$ make -j4
  DESCEND  objtool
  CALL    scripts/checksyscalls.sh
  CHK     include/generated/compile.h
  Building modules, stage 2.
Kernel: arch/x86/boot/bzImage is ready  (#1)
  MODPOST 5025 modules
```

If you have come this far without an error then Congratulations you have successfully added the system call into your kernel. The following steps are about how you can test that out

STEP 12:

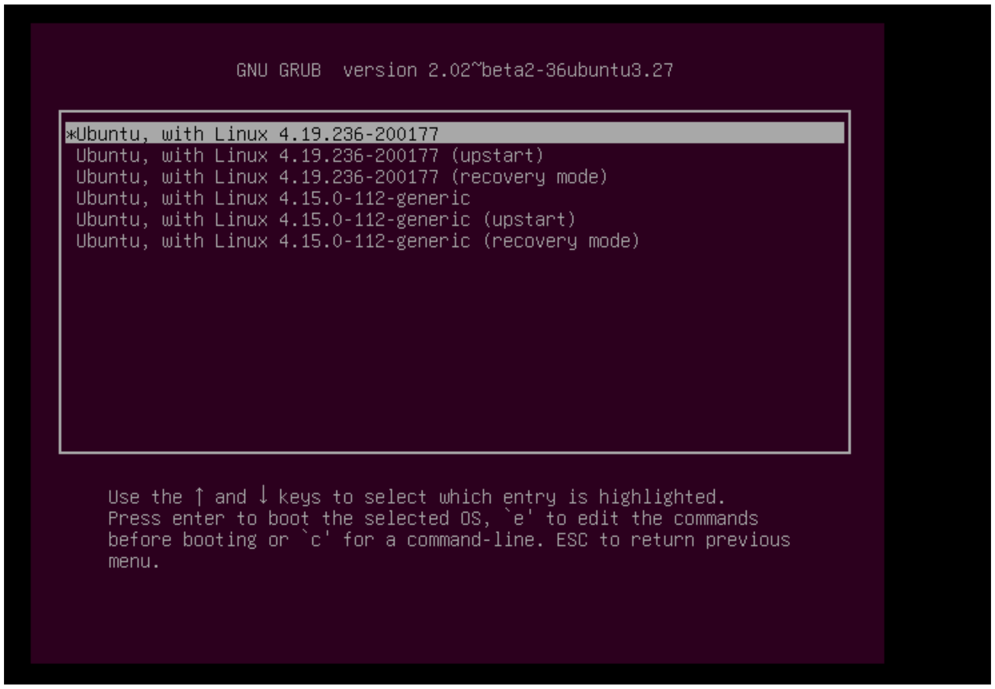
Now login as super user by typing "sudo su" and run the following command "make modules_install install"



```
root@ubuntu: /home/ammar/Downloads/linux-4.19.236
ammar@ubuntu:~/Downloads/linux-4.19.236$ sudo su
[sudo] password for ammar:
root@ubuntu:/home/ammar/Downloads/linux-4.19.236# make moduules_install install
make: *** No rule to make target 'moduules_install'.  Stop.
root@ubuntu:/home/ammar/Downloads/linux-4.19.236# make modules_install install
```

STEP 13:

Then write command "shutdown -r now" to restart your Virtual machine and hold Shift key to open the Grub Menu as shown below:



```
GNU GRUB  version 2.02~beta2-36ubuntu3.27

*Ubuntu, with Linux 4.19.236-200177
 Ubuntu, with Linux 4.19.236-200177 (upstart)
 Ubuntu, with Linux 4.19.236-200177 (recovery mode)
 Ubuntu, with Linux 4.15.0-112-generic
 Ubuntu, with Linux 4.15.0-112-generic (upstart)
 Ubuntu, with Linux 4.15.0-112-generic (recovery mode)

   Use the ↑ and ↓ keys to select which entry is highlighted.
   Press enter to boot the selected OS, `e' to edit the commands
   before booting or `c' for a command-line. ESC to return previous
   menu.
```

STEP 14:

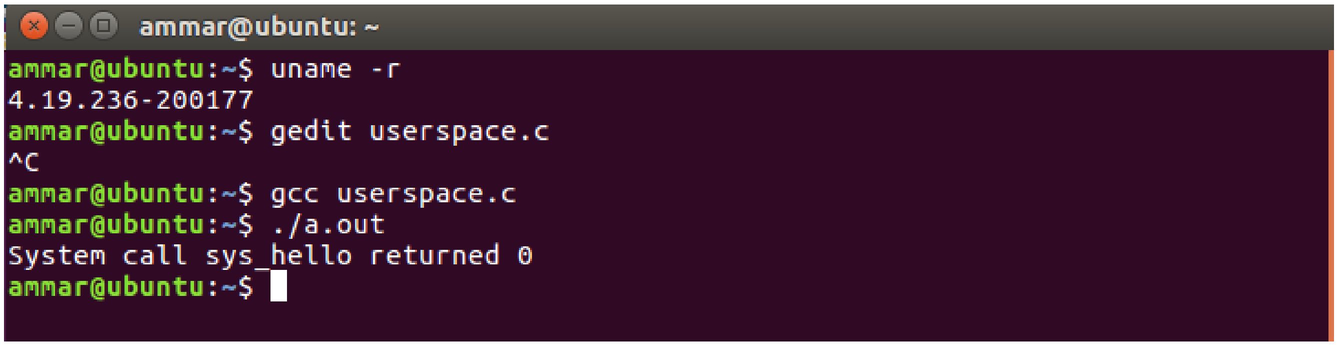
Now you can check your kernel version by typing “uname -r” command. After that make a file named userspace.c by using command “gedit userspace.c” and write down the following code in it:

```
#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;
}
```

(335 is the number we used in our syscalls table. Make sure it is same and correct)

STEP 15:

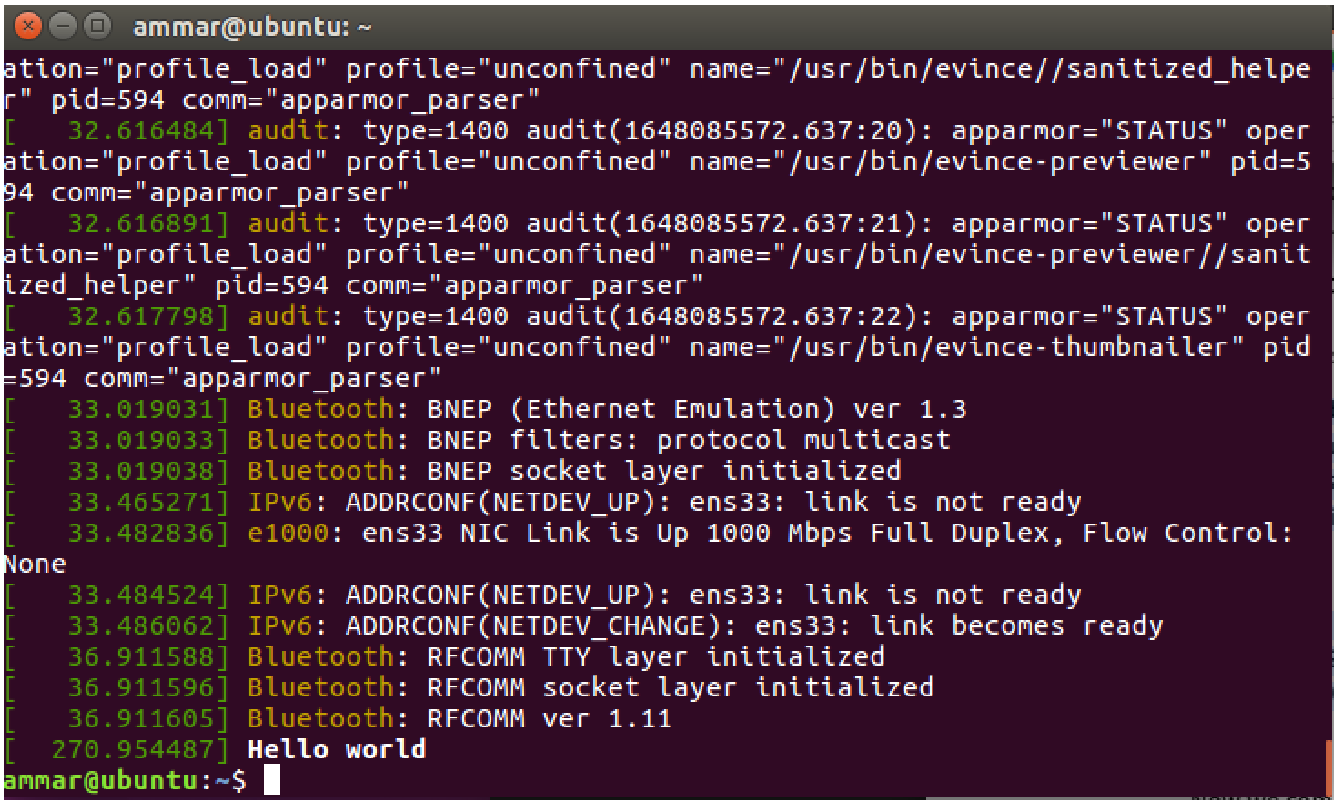
After this compile the file using "gcc userspace.c" and execute it using "./a.out" command on the kernel as shown below.



```
ammar@ubuntu:~$ uname -r
4.19.236-200177
ammar@ubuntu:~$ gedit userspace.c
^C
ammar@ubuntu:~$ gcc userspace.c
ammar@ubuntu:~$ ./a.out
System call sys_hello returned 0
ammar@ubuntu:~$ 
```

STEP 16:

You can also check your added system call using "dmesg" command on the terminal



```
ation="profile_load" profile="unconfined" name="/usr/bin/evince//sanitized_helpe
r" pid=594 comm="apparmor_parser"
[   32.616484] audit: type=1400 audit(1648085572.637:20): apparmor="STATUS" oper
ation="profile_load" profile="unconfined" name="/usr/bin/evince-previewer" pid=5
94 comm="apparmor_parser"
[   32.616891] audit: type=1400 audit(1648085572.637:21): apparmor="STATUS" oper
ation="profile_load" profile="unconfined" name="/usr/bin/evince-previewer//sanit
ized_helper" pid=594 comm="apparmor_parser"
[   32.617798] audit: type=1400 audit(1648085572.637:22): apparmor="STATUS" oper
ation="profile_load" profile="unconfined" name="/usr/bin/evince-thumbnailer" pid
=594 comm="apparmor_parser"
[   33.019031] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
[   33.019033] Bluetooth: BNEP filters: protocol multicast
[   33.019038] Bluetooth: BNEP socket layer initialized
[   33.465271] IPv6: ADDRCONF(NETDEV_UP): ens33: link is not ready
[   33.482836] e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Control:
None
[   33.484524] IPv6: ADDRCONF(NETDEV_UP): ens33: link is not ready
[   33.486062] IPv6: ADDRCONF(NETDEV_CHANGE): ens33: link becomes ready
[   36.911588] Bluetooth: RFCOMM TTY layer initialized
[   36.911596] Bluetooth: RFCOMM socket layer initialized
[   36.911605] Bluetooth: RFCOMM ver 1.11
[  270.954487] Hello world
ammar@ubuntu:~$ 
```