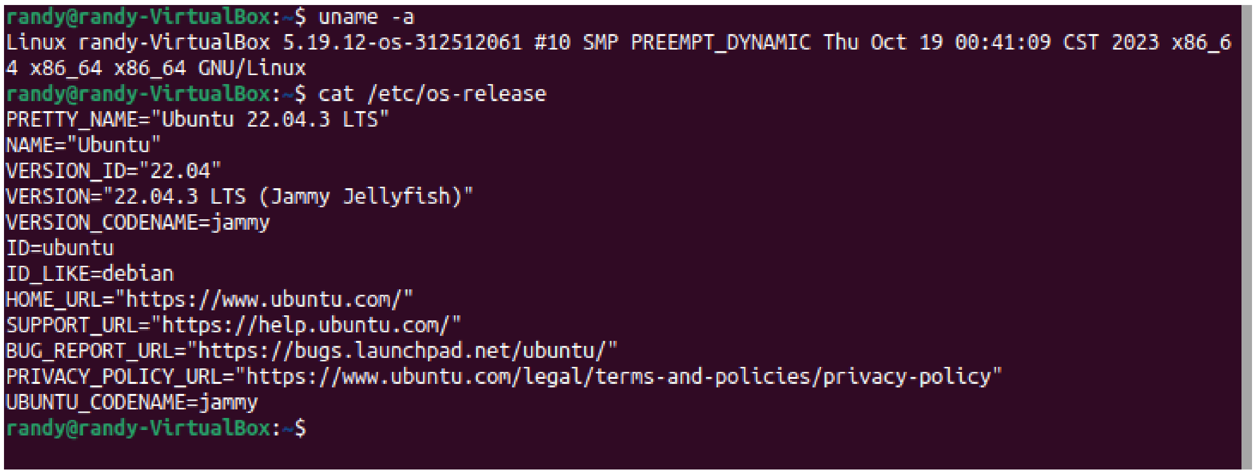
Assignment 1: Compiling Linux Kernel and Adding Custom System Calls Report

Kernel compilation

An example of a successful kernel compilation by showing the results of `uname -a` and `cat /etc/os-release` commands:



```
randy@randy-VirtualBox:~$ uname -a
Linux randy-VirtualBox 5.19.12-os-312512061 #10 SMP PREEMPT_DYNAMIC Thu Oct 19 00:41:09 CST 2023 x86_6
4 x86_64 x86_64 GNU/Linux
randy@randy-VirtualBox:~$ cat /etc/os-release
PRETTY_NAME="Ubuntu 22.04.3 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="22.04.3 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=jammy
randy@randy-VirtualBox:~$
```

Adding Custom System Calls

Implement Sys_hello & Sys_revstr

1. change the directory to where the files are extracted

`cd /usr/src/linux-5.19.12/`

2. Create a directory named `hello/` and change the directory to hello/:

```
mkdir hello
cd hello
```

Create a file `hello.c` using text editor:

`gedit hello.c`

3. write the following code in the editor:

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

asmlinkage long __x64_sys_hello(void)
{
        printk("Hello world\n");
        printk("312512061\n");
        return 0;
}
```

Create a file `revstr.c` using text editor:

`gedit revstr.c`

```
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/uaccess.h>
#include <linux/linkage.h>
SYSCALL_DEFINE2(revstr, int, len_count, char __user *, src) {
    char buf[256];
    unsigned long chunklen = sizeof(buf);
    int i;
    char temp;
    unsigned long len_max = len_count;
    if (len_max <chunklen)
      chunklen = len_max+1;
    if (copy_from_user(buf,src,chunklen))
    {
      return -EFAULT;
    }
    printk("The origin string:%s\n",buf);

    for(i=0;i<len_count/2;i++)
    {
      temp = buf[i];
      buf[i] = buf[len_count-i-1];
      buf[len_count-i-1] = temp;
    }
    printk("The reversed string:%s\n",buf);
    return 0;
}
```

printk prints to the kernel’s log file.

4. Create a “Makefile” in the hello directory:

   `gedit Makefile`

   and add the following line to it:

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

   This is to ensure that the hello.c file is compiled and included in the kernel source code.

5. Adding hello/ to the kernel’s Makefile:

   Go back to the parent dir i.e. `cd ../` and open “Makefile”

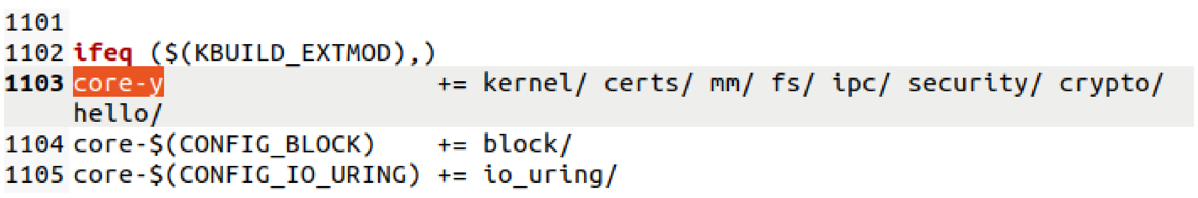
   `gedit Makefile`

   search for core-y in the document, you’ll find this line as the second instance of your search:

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

   Add ‘hello/’ to the end of this line:

   `core-y += kernel/ mm/ fs/ ipc/ security/ crypto/ hello/`



```

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

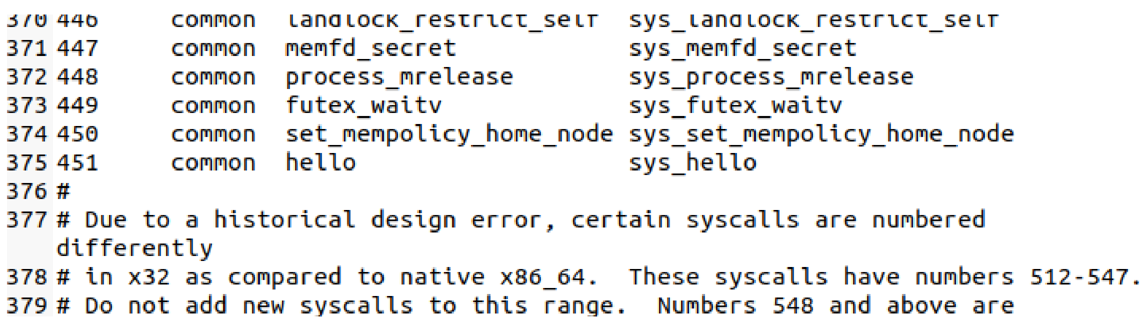
This is to tell the compiler that the source files of our new system call (sys_hello()) are in present in the hello directory.

6. Add the new system call to the system call table:

   Run the following commands in your terminal from linux-5.19.12/ directory:

```
cd arch/x86/entry/syscalls/
gedit syscall_64.tbl
```

   You'll get a file like the following in your editor:



```
446     common  landlock_restrict_self  sys_landlock_restrict_self
447     common  memfd_secret            sys_memfd_secret
448     common  process_mrelease        sys_process_mrelease
449     common  futex_waitv             sys_futex_waitv
450     common  set_mempolicy_home_node sys_set_mempolicy_home_node
451     common  hello                   sys_hello
#
# Due to a historical design error, certain syscalls are numbered
differently
# in x32 as compared to native x86_64.  These syscalls have numbers 512-547.
# Do not add new syscalls to this range.  Numbers 548 and above are
```

   Go to the last of the common document and add a new line like so:

   `451        64          hello             sys_hello`

   `451        common      revstr            sys_revstr`

7. Add new system call to the system call header file:

   Go to the `linux-5.19.12/` directory and type the following commands:

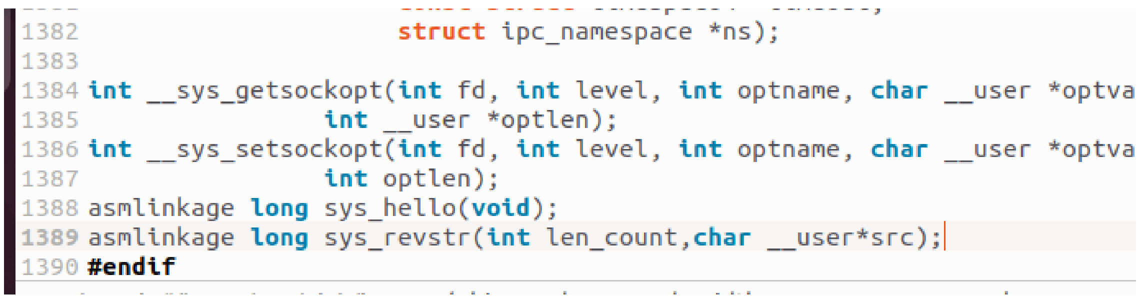
```
cd include/linux/
gedit syscalls.h
```

   Add the following line to the end of the document before the #endif statement:

   `asmlinkage long sys_hello(void);`

`asmlinkage long sys_revstr(int len_count,char __user*src);`

After this your file will look like so:



```
                          struct ipc_namespace *ns);

int __sys_getsockopt(int fd, int level, int optname, char __user *optva
                int __user *optlen);
int __sys_setsockopt(int fd, int level, int optname, char __user *optva
                int optlen);
asmlinkage long sys_hello(void);
asmlinkage long sys_revstr(int len_count,char __user*src);
#endif
```

Save and exit.

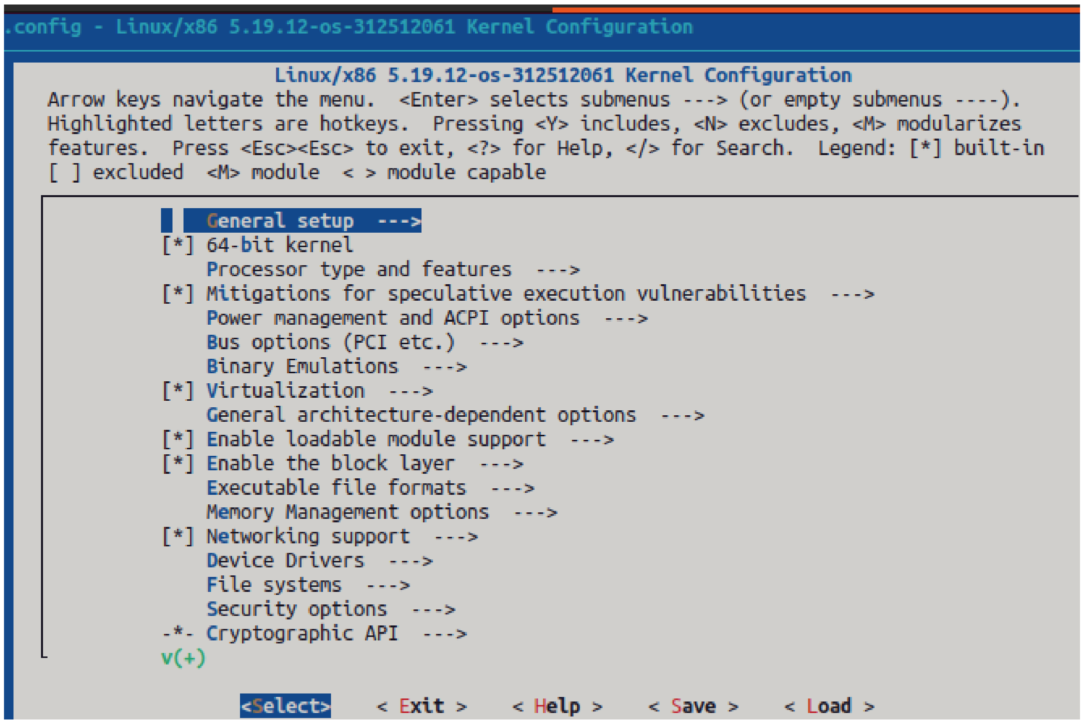
This defines the prototype of the function of our system call. “asmlinkage” is a key word used to indicate that all parameters of the function would be available on the stack.

8. Compile the kernel:

   to configure your kernel use the following command in your `linux-5.19.12/` directory:

   `sudo make menuconfig`

   Once the above command is used to configure the Linux kernel, you will get a pop up window with the list of menus and you can select “exit”



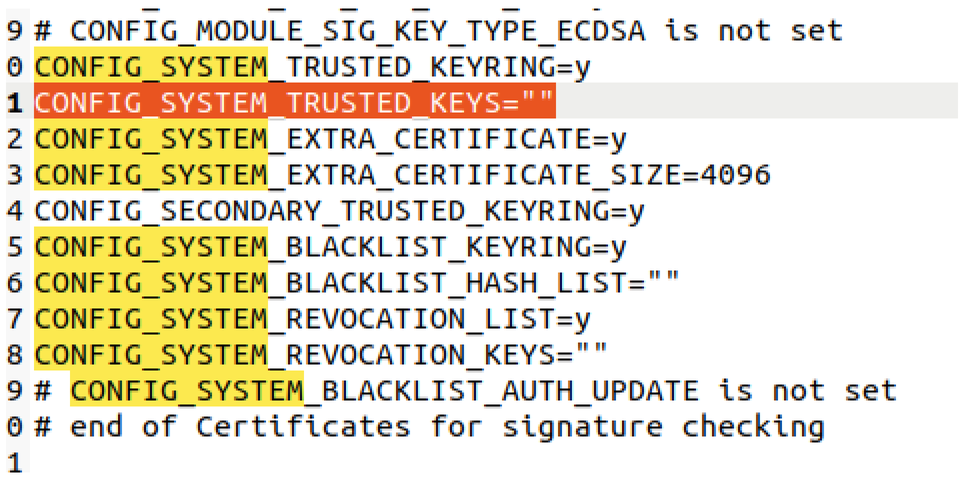
```
.config - Linux/x86 5.19.12-os-312512061 Kernel Configuration
                    Linux/x86 5.19.12-os-312512061 Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in
  [ ] excluded  <M> module  < > module capable

              General setup  --->
          [*] 64-bit kernel
              Processor type and features  --->
          [*] Mitigations for speculative execution vulnerabilities  --->
              Power management and ACPI options  --->
              Bus options (PCI etc.)  --->
              Binary Emulations  --->
          [*] Virtualization  --->
              General architecture-dependent options  --->
          [*] Enable loadable module support  --->
          [*] Enable the block layer  --->
              Executable file formats  --->
              Memory Management options  --->
          [*] Networking support  --->
              Device Drivers  --->
              File systems  --->
              Security options  --->
          -*- Cryptographic API  --->
          v(+)

              <Select>    < Exit >    < Help >    < Save >    < Load >
```

then `gedit .config`

let

CONFIG_SYSTEM_REVOCATION_KEYS=""

CONFIG_SYSTEM_TRUSTED_KEYS=""

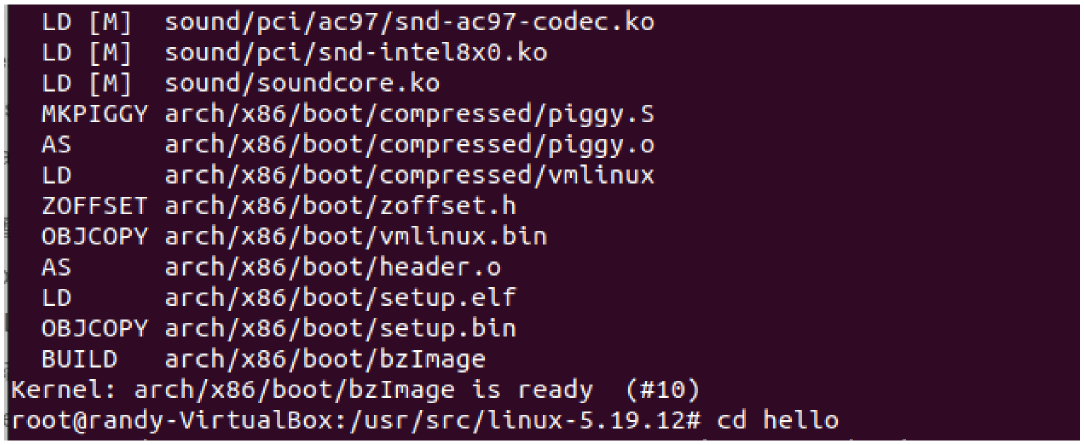


```
# CONFIG_MODULE_SIG_KEY_TYPE_ECDSA is not set
CONFIG_SYSTEM_TRUSTED_KEYRING=y
CONFIG_SYSTEM_TRUSTED_KEYS=""
CONFIG_SYSTEM_EXTRA_CERTIFICATE=y
CONFIG_SYSTEM_EXTRA_CERTIFICATE_SIZE=4096
CONFIG_SECONDARY_TRUSTED_KEYRING=y
CONFIG_SYSTEM_BLACKLIST_KEYRING=y
CONFIG_SYSTEM_BLACKLIST_HASH_LIST=""
CONFIG_SYSTEM_REVOCATION_LIST=y
CONFIG_SYSTEM_REVOCATION_KEYS=""
# CONFIG_SYSTEM_BLACKLIST_AUTH_UPDATE is not set
# end of Certificates for signature checking
```

Now to compile the kernel you can use the make command:

`sudo make -jn`

n is the number of core



```
  LD [M]  sound/pci/ac97/snd-ac97-codec.ko
  LD [M]  sound/pci/snd-intel8x0.ko
  LD [M]  sound/soundcore.ko
  MKPIGGY arch/x86/boot/compressed/piggy.S
  AS      arch/x86/boot/compressed/piggy.o
  LD      arch/x86/boot/compressed/vmlinux
  ZOFFSET arch/x86/boot/zoffset.h
  OBJCOPY arch/x86/boot/vmlinux.bin
  AS      arch/x86/boot/header.o
  LD      arch/x86/boot/setup.elf
  OBJCOPY arch/x86/boot/setup.bin
  BUILD   arch/x86/boot/bzImage
Kernel: arch/x86/boot/bzImage is ready  (#10)
root@randy-VirtualBox:/usr/src/linux-5.19.12# cd hello
```

After compiling, message show“Kernel: arch/x86/boot/bzImage is ready”

9. Install / update Kernel:

Run the following command in your terminal:

`sudo make modules_install install`

It will create some files under `/boot/`
directory and it will automatically make a entry in your grub.cfg. To check whether it made correct entry, check the files under `/boot/` directory . If you have followed the steps without any error you will find the following files in it in addition to others.

Now to update the kernel in your system reboot the system . You can use the following command:

`shutdown -r now`

Testing

- sys_hello

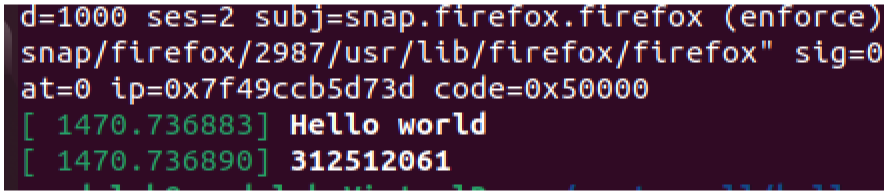
```
#include <assert.h>
#include <unistd.h>
#include <sys/syscall.h>

/*
 * 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 451

int main(int argc, char *argv[]) {
    int ret = syscall(__NR_hello);
    assert(ret == 0);

    return 0;
}
```

- sys_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 452

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;
}
```

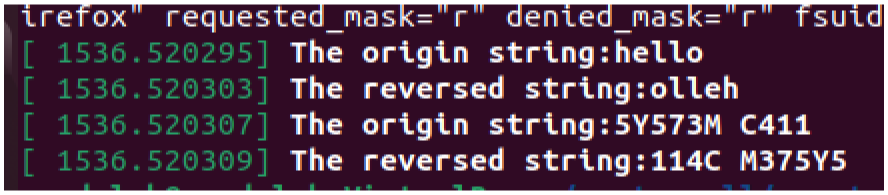
Result

- sys_hello



```
d=1000 ses=2 subj=snap.firefox.firefox (enforce)
snap/firefox/2987/usr/lib/firefox/firefox" sig=0
at=0 ip=0x7f49ccb5d73d code=0x50000
[ 1470.736883] Hello world
[ 1470.736890] 312512061
```

- sys_revstr



```
irefox" requested_mask="r" denied_mask="r" fsuid
[ 1536.520295] The origin string:hello
[ 1536.520303] The reversed string:olleh
[ 1536.520307] The origin string:5Y573M C411
[ 1536.520309] The reversed string:114C M375Y5
```