OS Assignment 1

Kernel Compilation

```
jason@jason-virtual-machine: /
jason@jason-virtual-machine:/$ uname -a
Linux jason-virtual-machine 5.19.12-os-311551137 #1 SMP PREEMPT_DYNAMIC Mon Oct
9 22:21:21 CST 2023 x86_64 x86_64 x86_64 GNU/Linux
jason@jason-virtual-machine:/$ 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-poli
cy"
UBUNTU_CODENAME=jammy
jason@jason-virtual-machine:/$
```

System Call

Steps for `sys_hello` and `sys_revstr` implementation:

1. `cd ~/linux-5.19.12`
2. `mkdir mysyscall`
3. `vim mysyscall/hello.c` and add the codes for `sys_hello` .

```c
hello.c > ...
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/syscalls.h>

SYSCALL_DEFINE0(hello)
{
    printk(KERN_INFO "Hello, world!\n");
    printk(KERN_INFO "311551137\n");
    return 0;
}
```

4. `vim mysyscall/revstr.c` and add the codes for `sys_revstr` .

revstr.c > ...

```c
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/uaccess.h>

SYSCALL_DEFINE2(revstr, int, length, const char __user *, str)
{
    char *oribuf;
    char *revbuf;
    int i;

    oribuf = kmalloc(length, GFP_KERNEL);
    if (!oribuf)
    {
        return -ENOMEM;
    }

    if (copy_from_user(oribuf, str, length))
    {
        kfree(oribuf);
        return -EFAULT;
    }

    oribuf[length] = '\0';

    revbuf = kmalloc(length, GFP_KERNEL);
    if (!revbuf)
    {
        return -ENOMEM;
    }

    for (i = 0; i < length; i++)
    {
        revbuf[length-1-i] = oribuf[i];
    }
    revbuf[length] = '\0';

    printk(KERN_INFO "The origin string: %s\n", oribuf);
    printk(KERN_INFO "The reversed string: %s\n", revbuf);

    kfree(oribuf);
    kfree(revbuf);
    return 0;
}
```

5. To link our codes to the kernel, `vim mysyscall/Makefile` and add the following content.

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

6. To help the kernel to locate our codes, `vim Makefile` and append `mysyscall/` after `core-y += ...` , i.e.,

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

becomes

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

7. In the end of `include/linux/syscalls.h` (before `#endif`), add the function prototype for the new entry point.

```
asmlinkage long sys_hello(void);
asmlinkage long sys_revstr(int length, char __user *str);
```

8. In `include/uapi/asm-generic/unistd.h`, the generic list, add the following entries.

```
#define __NR_hello 548
__SYSCALL(__NR_hello, sys_hello)

#define __NR_revstr 549
__SYSCALL(__NR_revstr, sys_revstr)
```

Also, increment `__NR_syscalls` by 2.

9. In `kernel/sys_ni.c`, which provides a fallback stub implementation of each system call, add

```
COND_SYSCALL(hello);
COND_SYSCALL(revstr);
```

10. Finally we need to update the master syscall tables. In the end of `arch/x86/entry/syscalls/syscall_64.tbl` add

```
548     common  hello                   sys_hello
549     common  revstr                  sys_revstr
```

11. Now we recompile the kernel

```
cd ~/linux-5.19.12 && make -j 4
// wait for compilation...

sudo make modules_install install -j 4
// wait for installation...

reboot
```

12. Test the given program in the assignment. They should work successfully.

The screenshot of the messages

- `sys_hello` :

```
[  285.069758] Hello, world!
[  285.069800] 311551137
```

- `sys_revstr` :

```
[  170.289974] The origin string: hello
[  170.289980] The reversed string: olleh
[  170.289993] The origin string: 5Y573M C411
[  170.289994] The reversed string: 114C M375Y5
```