**Part 1: Kernel Compilation**

In part 1, we need to change the kernel version to “-os-id”. The current Linux kernel .config was copied as our kernel, then the CONFIG\_LOCAL\_VERSION was modified as requested. Also, the KEYS were changed and Linux headers need to be reinstalled again. After kernel made and restart, the result should be shown as below Figure 1.

1. After unzip the downloaded kernel to corresponding directory, change authorizes to prevent authorized problem, where my username is don.

chown -R don:don linux-5.19.12

1. Make config file, copy current Linux release(uname -r) config file to our kernel build directory

cd linux-5.19.12/

make mrproper

cp -v /boot/config-$(uname -r) .config

1. Make config file

make menuconfig

1. Change version, CONFIG\_LOCALVERSION to change suffix, CONFIG\_SYSTEM\_TRUSTED\_KEYS a PEM-encoded file for additional certificates, CONFIG\_SYSTEM\_REVOCATION\_KEYS include X.509 certificates

vim .config

/\<CONFIG\_LOCALVERSION\>

# change "" to "-os-id"

CONFIG\_LOCALVERSION = "-os-id"

CONFIG\_SYSTEM\_TRUSTED\_KEYS=""

CONFIG\_SYSTEM\_REVOCATION\_KEYS=""

1. Check version before execute, CONFIG\_LOCALVERSION\_AUTO=y means add suffix

sudo apt install --reinstall linux-headers-$(uname -r)

make oldconfig && make prepare

make CONFIG\_LOCALVERSION\_AUTO=y kernelrelease

1. Install kernel and modules

make -j 4

sudo make modules\_install

sudo make install

1. Install grub properly to prevent boot issue

sudo grub-install /dev/sda

sudo update-grub

1. Restart and check version

uname -a

cat /etc/os-release

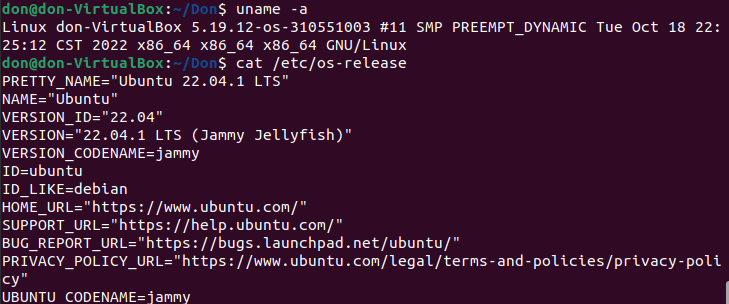


Figure 1 kernel compilation part – screenshot of the results

**Part 2: System Call**

In this part, we would like to add 2 new system calls to our kernel.

**Task 1: add print “Hello, world!” and student ID “310551003” system call to kernel.**

1. Restart computer and go to original Linux kernel and we can modify our own kernel.
2. As provided sample C code, the new system call “\_\_NR\_hello” was defined to 548. Looking into “linux-5.19.12/arch/x86/entry/syscalls/syscall\_64.tbl” which is Linux System Call Table contains all system call number and their corresponding system call function name. That is, the new system call is also need to be added on this table so that the system call number 548 will match to corresponding system call function.

vim ~Don/kernelbuild/linux-5.19.12/arch/x86/entry/syscalls/syscall\_64.tbl

548 common hello sys\_hello

1. Add system call function “asmlinkage long sys\_hello(void);” at system call header file “linux-5.19.12/include/linux/syscalls.h”

vim include/linux/syscalls.h

asmlinkage long sys\_hello(void);

1. Add system call function code, in corresponding kernel file. In my case, a new hello folder was created and hello.c code was created and the content was shown as below:

Where DEFINE0 mean 0 parameter will be requested to provide, and printk is the print function used in Linux kernel, we add “Hello, world!” and my student ID “310551003” accordingly, so that, when sys\_hello was called, it will print these 2 strings accordingly.

cd ~/Don/kernelbuild/linux-5.19.12/

mkdir hello

vim hello/hello.c

#include <linux/kernel.h>

#include <linux/syscalls.h>

SYSCALL\_DEFINE0(hello)

{

printk("Hello, world!\n");

printk("310551003\n");

return 0;

}

1. Add a Makefile for hello.c script inside hello folder and input “obj-y :=hello.c” for compile hello.o to kernel when execute Make command.

vim hello/Makefile

obj-y := hello.o

1. Add “hello/” to “core-y” row from the kernel Makefile, thus, it will contain the new create hello folder when Linux kernel Make.

vim Makefile

/core-y

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

1. Install kernel and restart Linux to check on our kernel.

sudo apt install --reinstall linux-headers-$(uname -r)

make oldconfig && make prepare

make CONFIG\_LOCALVERSION\_AUTO=y kernelrelease

make -j 4

sudo make modules\_install

sudo make install

sudo grub-install /dev/sda

sudo update-grub

1. Use the provide sample code as below to test the system call

#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 548

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

int ret = syscall(\_\_NR\_hello);

assert(ret == 0);

return 0;}

1. The process of system call

Execute sample code call syscall 🡪 kernel looking into Linux System Call Table and find the system call function name 🡪 use system call function accordingly

1. Use sudo dmesg and check whether the result “Hello, world!” and “310551003” was shown as below Figure 2.



Figure 2 Result screenshot of system call printed messages

**Task 2: provide length of string and string, utilize system call to output string and its reverse string.**

1. Restart computer and go to original Linux kernel and we can modify our own kernel.
2. As provided sample C code, the new system call “\_\_NR\_revstr” was defined to 548. Looking into “linux-5.19.12/arch/x86/entry/syscalls/syscall\_64.tbl” which is Linux System Call Table contains all system call number and their corresponding system call function name. That is, the new system call is also need to be added on this table so that the system call number 548 will match to corresponding system call function.

vim ~Don/kernelbuild/linux-5.19.12/arch/x86/entry/syscalls/syscall\_64.tbl

549 common revstr sys\_revstr.c

1. Add system call function “asmlinkage long sys\_revstr(int str\_len, char \_\_user \*input\_str);” at system call header file “linux-5.19.12/include/linux/syscalls.h”

vim include/linux/syscalls.h

asmlinkage long sys\_revstr(int str\_len, char \_\_user \*input\_str);

1. Add system call function code, in corresponding kernel file. In my case, a new revstr folder was created and revstr.c code was created and the content was shown as below:

Where DEFINE2 mean 2 parameters will be requested to input, char buf is char array which I used to store the data from user space by using copy\_from\_user function and return -EFAULT if error, create a for loop for store the char array in origin sequence and reverse sequence. Add a null character ‘\0’ to the end of char array to imply the end of string while printing.

cd ~/Don/kernelbuild/linux-5.19.12/

mkdir revstr

vim revstr/revstr.c

#include <linux/kernel.h>

#include <linux/syscalls.h>

SYSCALL\_DEFINE2(revstr, int, len, char \_\_user \*, str)

{

char buf[256], nor[256], rev[256];

if(copy\_from\_user(buf, str, len)) return -EFAULT;

for(int i=len-1, j=0; i>=0 ; i--, j++){

rev[j] = buf[i];

nor[j] = buf[j];

printk("rev[j] nor[j]\n");

}

rev[len] = '\0';

nor[len] = '\0';

printk("The origin string: %s\n", nor);

printk("The reversed string: %s\n", rev);

return 0;

}

1. Add a Makefile for hello.c script inside hello folder and input “obj-y :=revstr.c” for compile revstr.o to kernel when execute Make command.

vim revstr/Makefile

obj-y := revstr.o

1. Add “revstr/” to “core-y” row from the kernel Makefile, thus, it will contain the new create hello folder when Linux kernel Make.

vim Makefile

/core-y

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

1. Install kernel and restart Linux to check on our kernel.

sudo apt install --reinstall linux-headers-$(uname -r)

make oldconfig && make prepare

make CONFIG\_LOCALVERSION\_AUTO=y kernelrelease

make -j 4

sudo make modules\_install

sudo make install

sudo grub-install /dev/sda

sudo update-grub

1. Use the provide sample code as below to test the system call

#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 549

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;

}

1. The process of system call

Execute sample code call syscall 🡪 kernel looking into Linux System Call Table and find the system call function name 🡪 use system call function accordingly

1. Use sudo dmesg and check whether the result origin and reversed string was shown as below Figure 3.





Figure 3 result of origin and reversed string

This homework is to familiar how to create a kernel, add and utilize system call from user space accordingly, understand the system call and user space relationship and implement it.