**Kennesaw State University**

**CS 3502 Operating Systems**

**Project 1 - System call**

Instructor: Kun Suo

Points Possible: 100

Difficulty: 

**Part A: Build the Linux kernel (50 points)**

**--- Please work this part on the VMs you built on your own laptop**



Create a virtual machine using VirtualBox or UTM (for Apple Silicon) on your laptop. As the kernel compiling is pretty large, please make sure your VM has at least 4GB memory and 80GB storage (if your disk is small, create a 60GB disk for the VM). For the Operating System, use Ubuntu 22.04 iso: <https://releases.ubuntu.com/jammy/>. There exist images for two architectures: x86 (e.g., Intel, AMD) and arm (e.g., Apple silicon). Please select the one that fits your machine.

How to build one Ubuntu VM?

Windows 10 (x86):

<https://www.youtube.com/watch?v=QbmRXJJKsvs>

MacOS (x86):

<https://www.youtube.com/watch?v=GDoCrfPma2k&t=321s>

MacOS (arm):

<https://youtu.be/O19mv1pe76M?si=4cYayFiqPNoHoY1w>

**Step 1: Get the Linux kernel code**

Before you download and compile the Linux kernel source, make sure you have development tools installed on your system. We recommend you work this project on your virtual machine.

In Ubuntu, install this software using apt:

$ sudo apt-get install -y gcc libncurses5-dev make wget flex bison vim libssl-dev libelf-dev

To obtain the version of your current kernel, type:

$ uname -r

5.15

For newer distributions of Ubuntu, you can see 5.x or 6.x. Please screenshot it and save it as the original kernel version.

Here is the screenshot of my current kernel:

A screenshot of a computer

Description automatically generated

Then, download kernel 5.19 and extract the source:

$ wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.19.tar.gz

$ tar xvzf linux-5.19.tar.gz

We will refer LINUX\_SOURCE to the top directory of the kernel source. Go to the linux source code folder:

$ cd linux-5.19

**Step 2: Configure your new kernel**

Before compiling the new kernel, a .config file needs to be generated in the top directory of the kernel source. To generate the config file and make possible changes to the default kernel configurations, type:

$ make localmodconfig

For Y/N, select all “N” if any questions on the terminal to minimize the configuration file. All other selection, just select the default one. (There could be tens or even hundreds Y/N to choose, just be patient, here is sample on my side: <https://youtu.be/hnROoXlTvcc?si=br2g1xb8iGESvx-S>)

Here, we avoid using $ make menuconfig to save the kernel compiling time. You can check .config using the following command under kernel folder. (<https://youtu.be/UyOGF4UOoR0>)

$ ls -al

Here is a screenshot of my VM:

A screenshot of a computer

Description automatically generated

*-------------------------------------------------[Possible Error] -------------------------------------------------*

*For some distributions of Ubuntu, you may see errors like this when compiling:*

*No rule to make target 'debian/canonical-certs.pem', needed by 'certs/x509\_certificate\_list'.*

*[Solution]*

*Edit the .config file and change the value of CONFIG\_SYSTEM\_TRUSTED\_KEYS to null*

$ vim .config

*Before:*

*CONFIG\_SYSTEM\_TRUSTED\_KEYRING=y*

*CONFIG\_SYSTEM\_TRUSTED\_KEYS="debian/canonical-certs.pem"*

*After:*

*CONFIG\_SYSTEM\_TRUSTED\_KEYRING=y*

*CONFIG\_SYSTEM\_TRUSTED\_KEYS=""*

*If the CONFIG\_SYSTEM\_REVOCATION\_KEYS="debian/canonical-revoked-certs.pem“ is not null, please also set it as null as:*

*CONFIG\_SYSTEM\_REVOCATION\_KEYS=""*

*Then recompile the kernel using the make command.*

*-----------------------------------------------------------------------------------------------------------------------*

**Step 3: Compile the kernel**

*Please keep in mind that the compiling might take 0.5-1 hour, depending on your machine hardware specs and speed. For instance, in my 2021 Macbook, it takes about 20 mins.*

In LINUX\_SOURCE, compile to create a compressed kernel image:

$ make

If your VM has more than 1 core, we suggest you use "make -j N" to accelerate the compiling. Here, N denotes the number of CPUs on your VM.

To compile kernel modules:

$ make modules

You can use "make modules -j N" to accelerate the compiling. Here N denotes the number of CPUs on your VM.

**Step 4: Install the kernel**

Install kernel modules (become a root user, use the su command):

$ sudo make modules\_install

Install the kernel:

$ sudo make install

If you are using Ubuntu, you need to create an init ramdisk manually:

$ sudo mkinitramfs -o /boot/initrd.img-5.19.0

$ sudo update-initramfs -c -k 5.19.0

The kernel image and other related files have been installed into the /boot directory. You can check it from /boot/grub/grub.cfg. Linux will boot by default using the 1st menu item.

**Step 5: Modify grub configuration file**

If you are using Ubuntu: change the grub configuration file:

$ sudo vim /etc/default/grub

Make the following changes:

GRUB\_DEFAULT=0

GRUB\_TIMEOUT=10

If your GRUB\_HIDDEN\_TIMEOUT\_QUIET=true, change it to GRUB\_HIDDEN\_TIMEOUT\_QUIET =false. If there is no GRUB\_HIDDEN\_TIMEOUT\_QUIET, just ignore it.

If your GRUB\_TIMEOUT\_STYLE=hidden, change it to GRUB\_TIMEOUT\_STYLE=menu. If there is no GRUB\_TIMEOUT\_STYLE, just ignore it.

Then, update the grub entry:

$ sudo update-grub2

**Step 6: Reboot your VM**

Reboot to the new kernel:

$ sudo reboot

*(If you are using university VM, ignore the following steps. It only works for local VM)*

Immediately after the BIOS/UEFI splash screen during boot, with BIOS, quickly press and hold the Shift key, which will bring up the GNU GRUB menu. (If you see the Ubuntu logo, you've missed the point where you can enter the GRUB menu.)

Select the following option:

A screenshot of a computer

Description automatically generated

Under the Advanced options, you will see the old kernel and the new kernel:

A screenshot of a computer

Description automatically generated

Select the new kernel and wait for a few seconds, you will enter the VM with new kernel:

A screenshot of a computer

Description automatically generated

After boot, check if you have the new kernel:

$ uname -r

5.19.0

Here is the screenshot of my new kernel:

A screenshot of a computer

Description automatically generated

**Submission of Part A:**

Please submit the screenshot of $uname -r in your old and new kernel.

**Part B: Add a new system call into the Linux kernel (50 points)**

**--- Please work this part on VMs in KSU cloud,** <https://cseview.kennesaw.edu/>

A diagram of a application

Description automatically generated

In this assignment, we add a simple system call helloworld to the Linux kernel. The system call prints out a hello world message to the syslog. You need to implement the system call in the kernel and write a user-level program to test your new system call.

Please note that the following only works on x86 VM, not on ARM VM. All VMs in KSU datacenter are x86 VMs.

**Step 1: Check the available system call number**

$ sudo find / -name unistd\_64.h



As my current Linux version is 5.4.0-150-generic, so I select the above one.

(Note: In the new version of the kernel ( >= 5.7 ), the kallsyms\_lookup\_name function is no longer exported for security reasons and cannot be directly used in kernel modules. Please boot your VM using the kernel <5.6.x)

Then, use cat command to print the file content:

$ sudo cat /usr/src/linux-headers-5.4.0-150-generic/arch/x86/include/generated/uapi/asm/unistd\_64.h

A screenshot of a computer

Description automatically generated

…

A black background with a black square

Description automatically generated with medium confidence

As we can see No.335 is not used yet, so we select 335 as our new system call.

**Step 2: Create a kernel module syscall**

$ vim syscall.c

Please note that the following only works in 64-bit system. The content of syscall.c is as follows:

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer program

Description automatically generated

//This is the system call No.335 body

A computer screen with white and green text

Description automatically generated

Source file: <https://github.com/kevinsuo/CS3502/blob/master/syscall.c>

**Step 3: Define the Makefile**

$ vim Makefile

A screenshot of a computer

Description automatically generated

Source file: <https://github.com/kevinsuo/CS3502/blob/master/Makefile>

**Step 4: Compile and enable the module syscall**

Under the directory with syscall.c and Makefile, run the make command to compile them.

$ sudo make

As the following figure shows, the red and blue parts are before and after the compiling.



Insert kernel modules into the Linux kernel

$ sudo insmod syscall.ko



You can use the $ lsmod to check the enabled modules:

A screenshot of a computer

Description automatically generated

If you want to disable one module, try $ sudo rmmod [mod-name].ko

**Step 5: write a test program to test your system call**

Create a test program test.c

A screen shot of a computer

Description automatically generated

Compile the user level program:

$ gcc test.c -o test.o

Test the new system call by running:

$ sudo ./test.o

The test program will call the new system call and output “Here is my syscall in OS kerenl!” message at the tail of the output of dmesg.

$ dmesg | grep my

Here is the screenshot on my VM:



**Submission of Part B:**

Please update the system call above so that your system call message will print out “Here is my syscall in the OS kernel by [Your Name]!”. Then, submit the screenshot of $ dmesg | grep “by Your name”.

For instance, a student named Sisi should upload a screenshot like:

