**Operating Systems Lab**

**Assignment 9**

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

**Name**: Viraj Sonawane

**Roll number**: 33382

**Batch**: N11

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

**Aim:**

Implement a new system call, add this new system call in the Linux kernel (any kernel source, any architecture and any Linux kernel distribution) and demonstrate the use of same.

**Objective:**

Add a new system call, swipe(), to the Linux kernel that transfers the remaining time slice of each process in a specified set to a target process. You will also demonstrate various uses of the system call (both advantageous and detrimental) **Theory:**

**Adding a simple system call:**

1. **Download the kernel source:**

In your terminal type the following command: wget https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.17.4.tar.xz Else go to kernel.org and download the latest version.

1. **Extract the kernel source code:**

sudo tar -xvf linux-4.17.4.tar.xz -C/usr/src/

tar — Tar stores and extracts files from a tape or disk archive.

-x — extract files from an archive

-v — requested using the –verbose option, when extracting archives

-f — file archive; use archive file or device archive

-C — extract to the directory specified after it.(in this case /usr/src/) Now, we’ll change the directory to where the files are extracted:

**3. Define a new system call sys\_hello( ):**

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

Write the following code in the editor:



**printk** prints to the kernel’s log file.

Create a “Makefile” in the hello directory:

**gedit Makefile** and add the following line to it:

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

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

Note: There is no space in between“obj-y”.

**4. 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/ block/ Add ‘hello/’ to the end of this line: core-y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ hello/

**Note: There is a space between “block/” and “hello/”. (Doing such a mistake may cause errors in further steps)**

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

**5. Add the new system call to the system call table:**

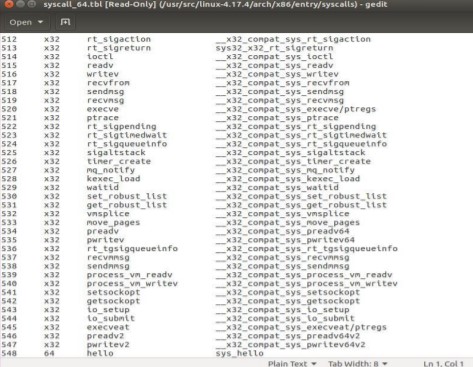
If you are on a 32-bit system you’ll need to change ‘syscall\_32.tbl’.

For 64-bit, change ‘syscall\_64.tbl’.

Run the following commands in your terminal from linux-4.17.4/ directory:

**cd arch/x86/entry/syscalls / gedit syscall\_64.tbl**

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

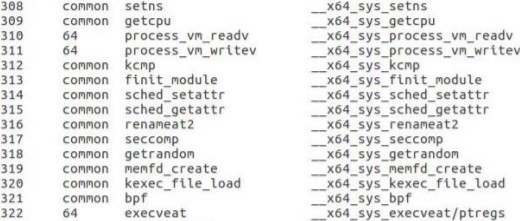


Go to the last of the document and add a new line like so:

**548 64 hello sys\_hello**

Note: Here 548 is written because in the previous line the number entry was 547. Remember this number it will be used in the later steps.

Also, note that I’ve written 64 in my system because it is 64 bit. You may have to write i586 or x32. For knowing what is to be written check in this file itself in many of the lines you may find entries like so:



64 written at 310, 311 and 322 line numbers.

This will tell you whether to write i586 or something else.

Save and exit.

**6. Add new system call to the system call header file:**

Go to the linux-4.17.4/ 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);**

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.

**7. Compile the kernel:**

Before starting to compile you need to install a few packages. Type the following commands in your terminal:

sudo apt-get install gcc

sudo apt-get install libncurses5-dev sudo apt-get ev sudo apt-get install bison sudo apt-get install flex sudo apt-get install libssl-dev sudo apt-get install libelf-dev sudo apt-get update sudo apt-get upgrade

to configure your kernel use the following command in your **linux-4.17.4/** 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 the items for the new configuration. If your unfamiliar with the configuration just check for the file systems menu and check whether “ext4” is chosen or not, if not select it and save the configuration.

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

**sudo make**

Pro Tip:

The make command can take a lot of time in compiling, to speed up the process you can take advantage of the multiple cores that our systems have these days. Simply type, **sudo make -jn** where n is the number of cores that you have in your linux system.

For example if you have a Quad core(4) processor, you can write:

**sudo make -j4** this will speed up my make process 4x times.

This might take an hours or more depending on your system.

**8. 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.

1. System.map-4.17.4
2. vmlinuz-4.17.4
3. initrd.img-4.17.4
4. config-4.17.4

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

**shutdown -r now**

After rebooting you can verify the kernel version using the following command:

**uname -r**

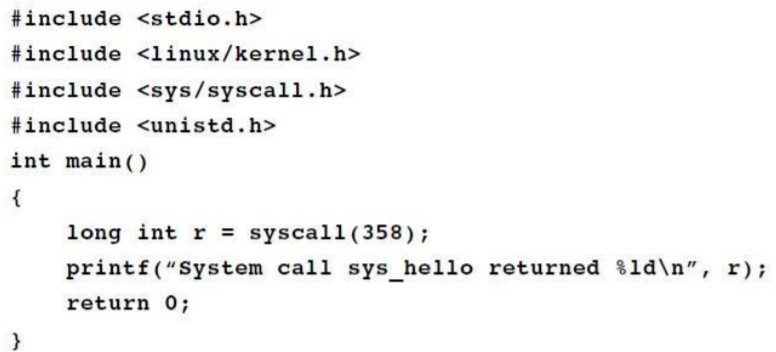
It will display the kernel version like so:

**4.17.4**

**9. Test system call:**

Go to your home(~) directory using the following commands and create a userspace.c file.

cd ~ gedit userspace.c



**Note: Remember to keep in mind the number of system call that is added in syscalls\_64.tbl? In my case the number was 548. Write that same number in your userspace.c file as an argument in syscall() function.**

Now, compile and run the program: gcc userspace.c

./a.out

If all the steps are done correctly you’ll get an output like below:

**System call sys\_hello returned 0**

Now, to check the message of your kernel run the following command: **dmesg**

This will display Hello world at the end of the kernel’s message.

Write the following code in this file:

**Conclusion:**

A new system call, add this new system call in the Linux kernel (any kernel source, any architecture, and any Linux kernel distribution) was studied.