AOS ASSIGNMENT 2

2021201086|Soumodipta Bose

ENVIRONMENT SETUP:

First setup a VM[Oracle Virtual Box] 20GB 4 Processors 4GB-RAM
Guest os: Ubuntu 20.04
Download the kernel from https://www.kernel.org/
Version : **4.19.210**

I have extracted the tar in **/home** as **/usr** did not have enough space for development.

```
sudo tar -xvf linux-4.19.210.tar.xz
```

I have referred to the following article as specified in the Assignment PDF.
https://medium.com/anubhav-shrimal/adding-a-hello-world-system-call-to-linux-kernel-dad32875872
I faced a permission issue during the make process and to solve it changed the following in the .config file:
CONFIG_SYSTEM_TRUSTED_KEYS="debian/canonical-certs.pem" to CONFIG_SYSTEM_TRUSTED_KEYS=""

LOCATIONS TO BE AWARE OF:

ROOT of all my subfolders:/home/soumodiptab/linux-4.19.210/
(so whenever i say '/' i mean the above location.)

- **include/linux/ : syscalls.h** (where the system call needs to be declared)
- **arch/x86/entry/syscalls/ : syscall_64.tbl** (contains the master syscalls table for my **x86_64** system architecture)
- **custom/:** newly created folder where I have all my **c files** which contains the implementation of all my system calls.

INCLUDING BUILD PATH:

I added my newly created directory /**custom/** to the Makefile build path.

```
986 ifeq ($(KBUILD_EXTMOD),)
987 core-y          += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ custom/
988
```

After all the c files are build to ensure that the object files get added to the kernel we have to add the object file names to the Makefile in our **/custom/** directory.

BUILD COMMANDS:

```
make -j8
make modules -j8 (only once)
make modules_install install -j8
```

*(jn - n being the number of threads allocated to **make** to speed up the process)*

STANDARD SYSCALLS(as per torvalds/linux):

I know we are supposed to make syscalls like **ayushihello,** but I have written all my syscalls in a standardized fashion as per the official linux kernel conventions and also maintained the naming convention of the assignment so all my system call entry points start with **sys_<my_name><sysname>** after macro expansion , I hope you can consider that.

I would also like to declare that I am using macros to define all my system calls using **SYSCALL_DEFINEx()** as it is specified in the official documentation:
https://www.kernel.org/doc/html/latest/process/adding-syscalls.html

Question 1:

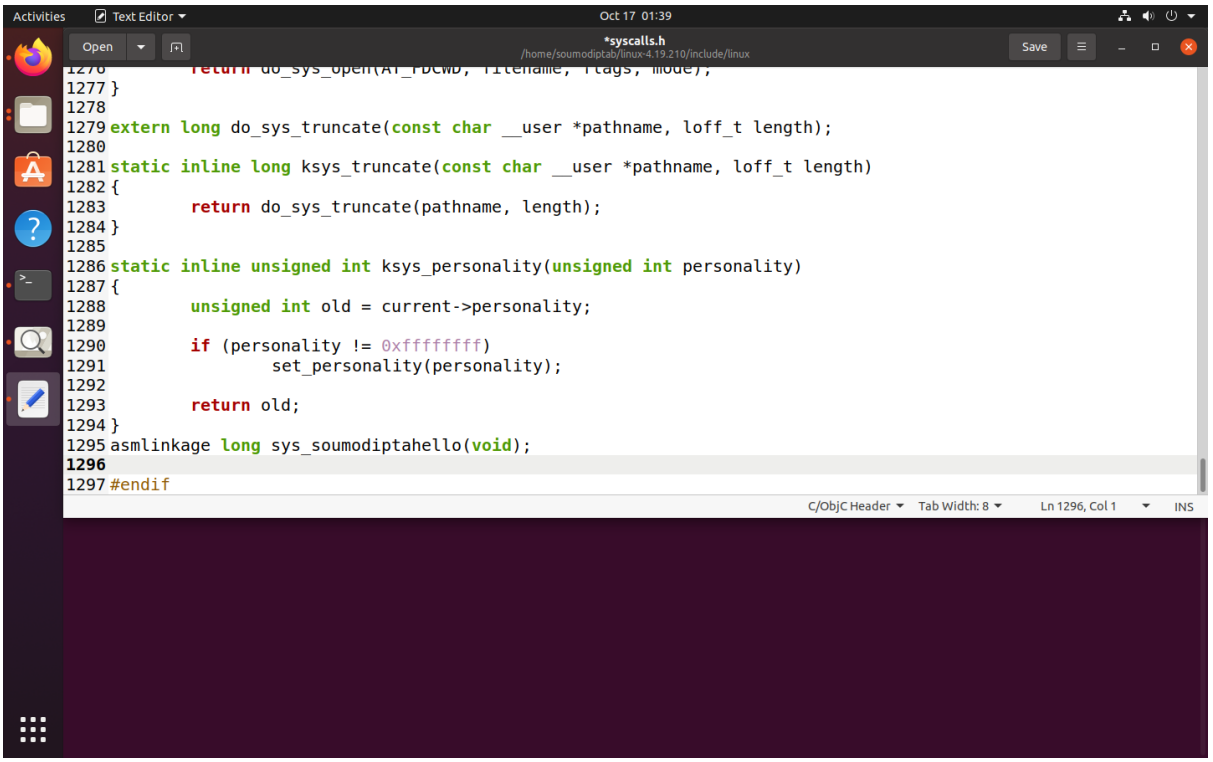
Write syscall to print a welcome message to Linux logs.

e.g., If your name is ayushi mahajan then syscall name will be ayushihello() Which will print some hello message to Linux logs upon calling it.

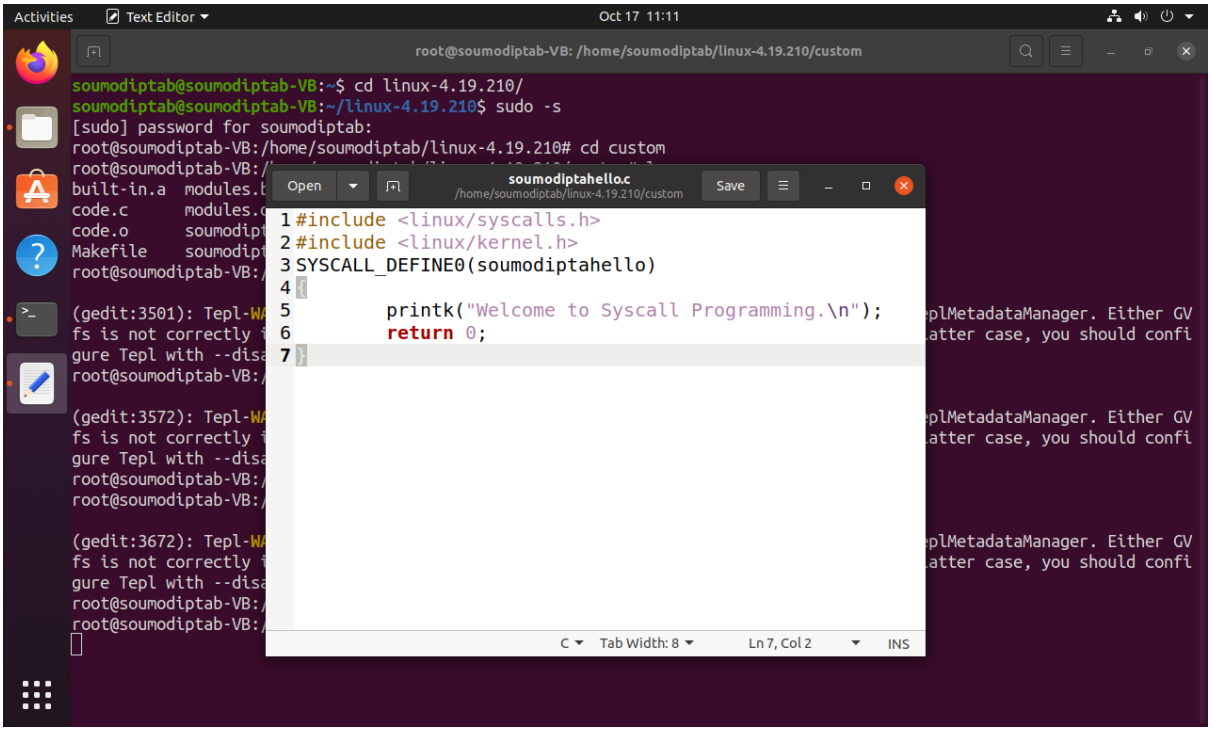
Solution:

Steps:

1. Traverse to **/include/linux/** and edit **syscalls.h** and add the definition of our system call.

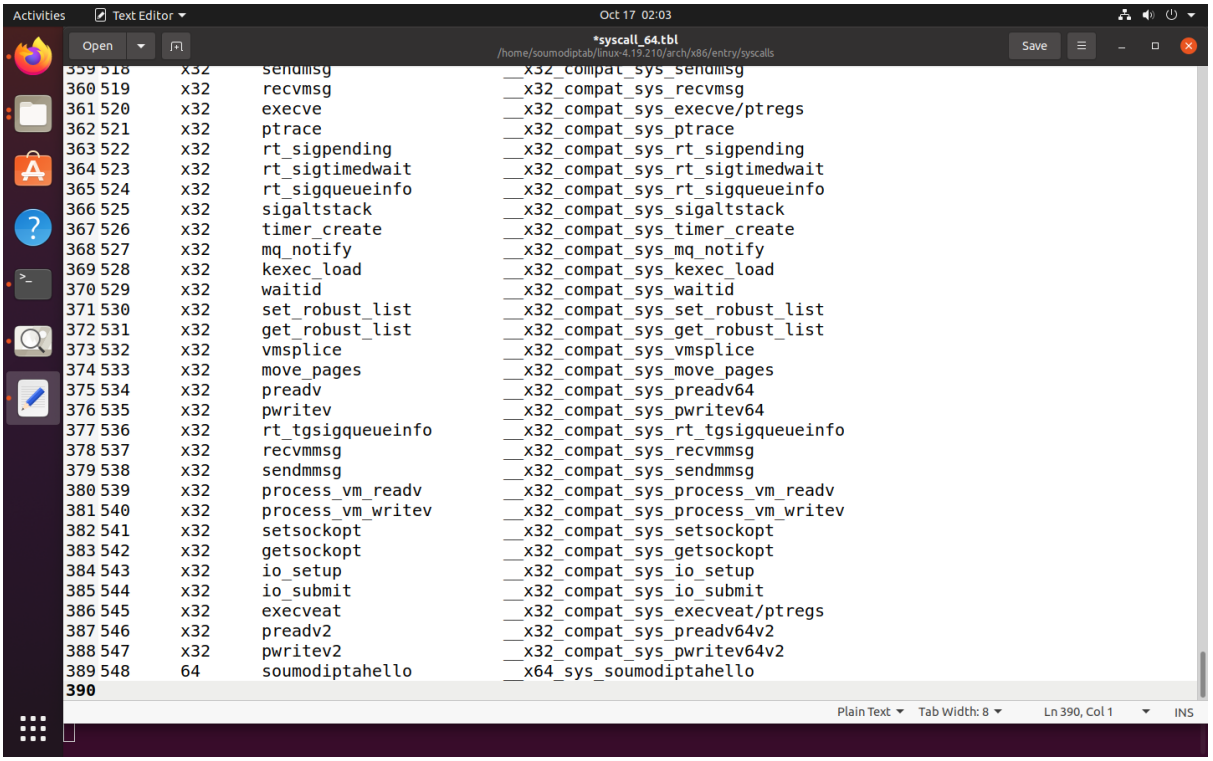


2. Traverse to **/custom/** and create a file using gedit: **soumodiptahello.c**
3. Create a new system call called: **soumodiptahello** using standard linux macros.



I am using the macro-expansion from **linux/syscalls.h** which uses **SYSCALL_DEFINEx** to define the number and type of arguments to the system call , it automatically generates the syscall stub into something like **sys_soumodiptahello(void)**,i am using the same in my other syscalls as well.

4. Add **soumodiptahello.o** in the Makefile so that it gets added to the build path.
5. Now traverse to **/arch/x86/entry/syscalls/** and add a syscall entry in the table **syscall_64.tbl**



6. Traverse back to the root and use the **build** commands to compile the kernel and install them.
7. Reboot the system and load the kernel from **GRUB** menu.

8. For Testing our newly added system call I have used a driver code to call that system call.

Open ▾ q1.c ~/syscalls

```c
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>
int main()
{
        long int ret_status = syscall(548);
        printf("System call soumodiptahello returned %ld\n", ret_status);
        return 0;
}
```

9. Now checking in kernel logs using **dmesg**:

```
root@soumodiptab-VB: /home/soumodiptab/syscalls
root@soumodiptab-VB:/home/soumodiptab/syscalls# ./q1
System call soumodiptahello returned 0
root@soumodiptab-VB:/home/soumodiptab/syscalls# dmesg
[ 2734.090104] Welcome to Syscall Programming.
root@soumodiptab-VB:/home/soumodiptab/syscalls#
```

(Note: i previously cleared all the dmesg logs using **dmesg -c**)

OBSERVATIONS:

- The linux kernel maintains logs for each activity that goes on inside the kernel and also there is a priority level associated with it like **KERN_EMERG**, **KERN_ALERT**, **KERN_CRIT**, **KERN_ERROR** and depending on this the kernel prints in the logs using the **printk** function.

Question 2:

Write syscall which will receive string parameter and print it along with some message to kernel logs.

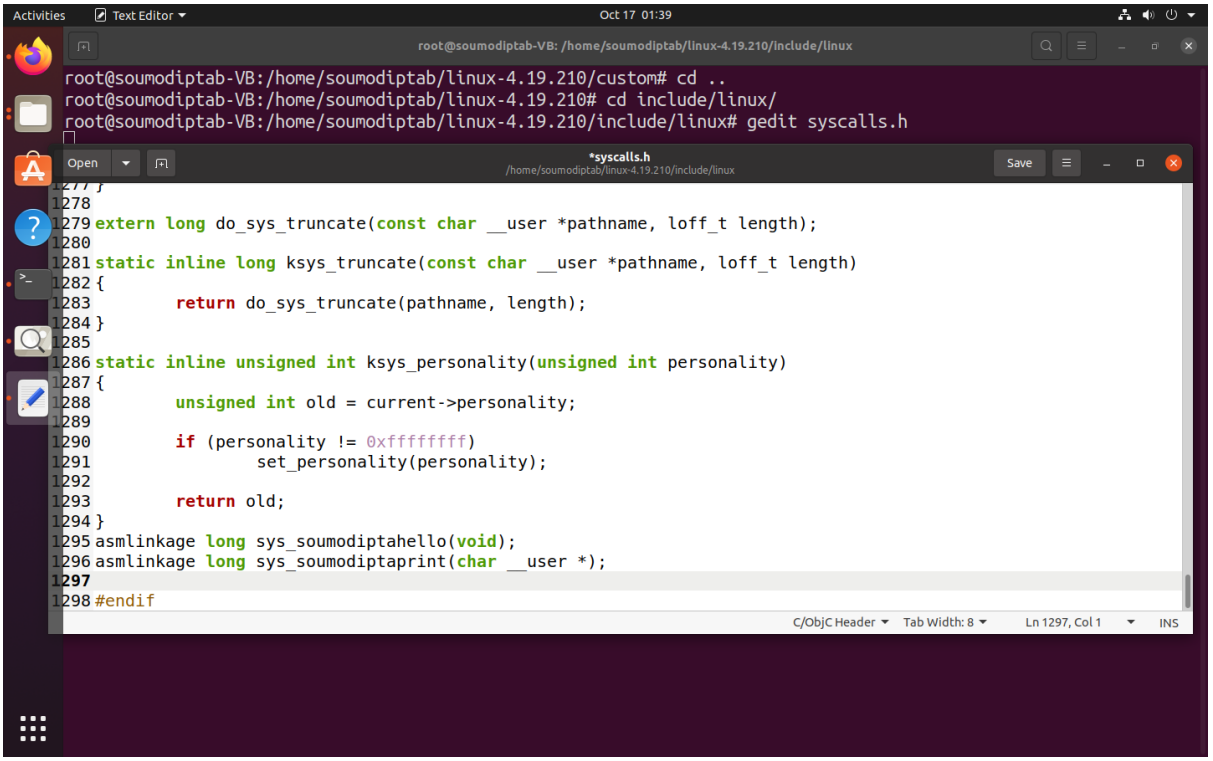
e.g., If your name is ayushi mahajan then syscall name will be ayushiprint(string)

Which will print passed parameters along with some message to Linux logs.

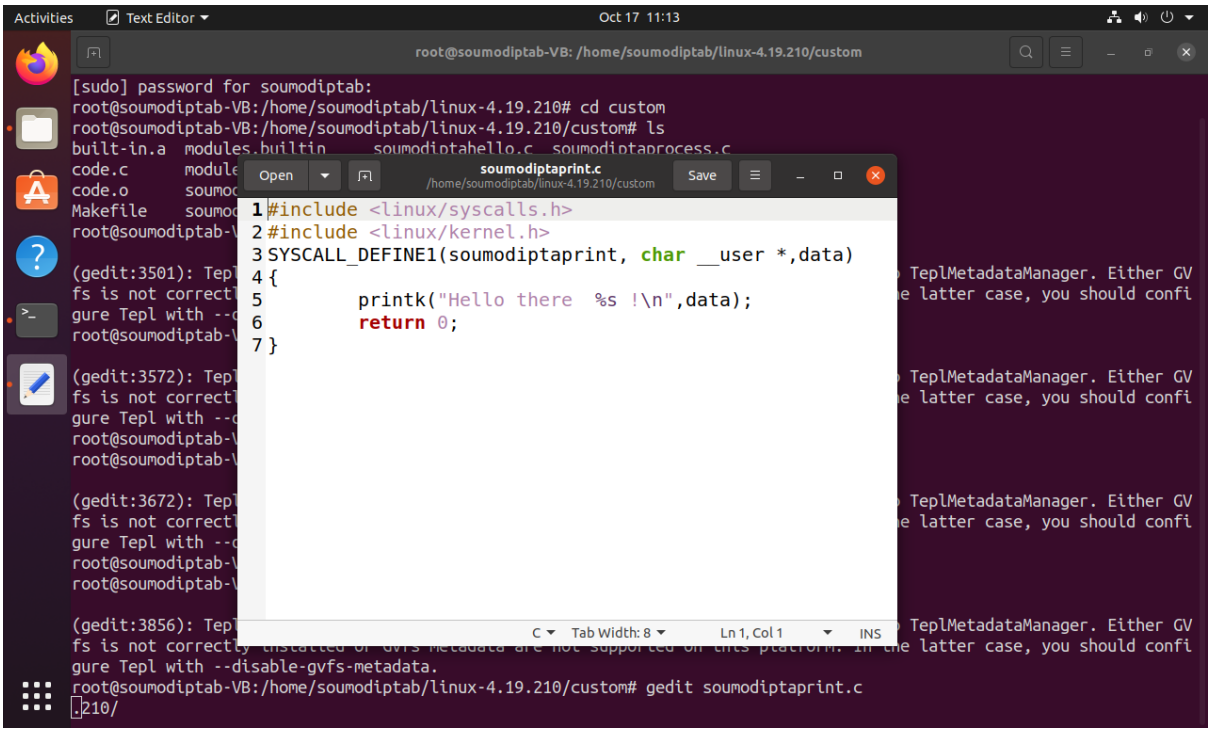
Solution:

Steps:

1. Traverse to **/include/linux/** and edit **syscalls.h** and add the definition of the system call.



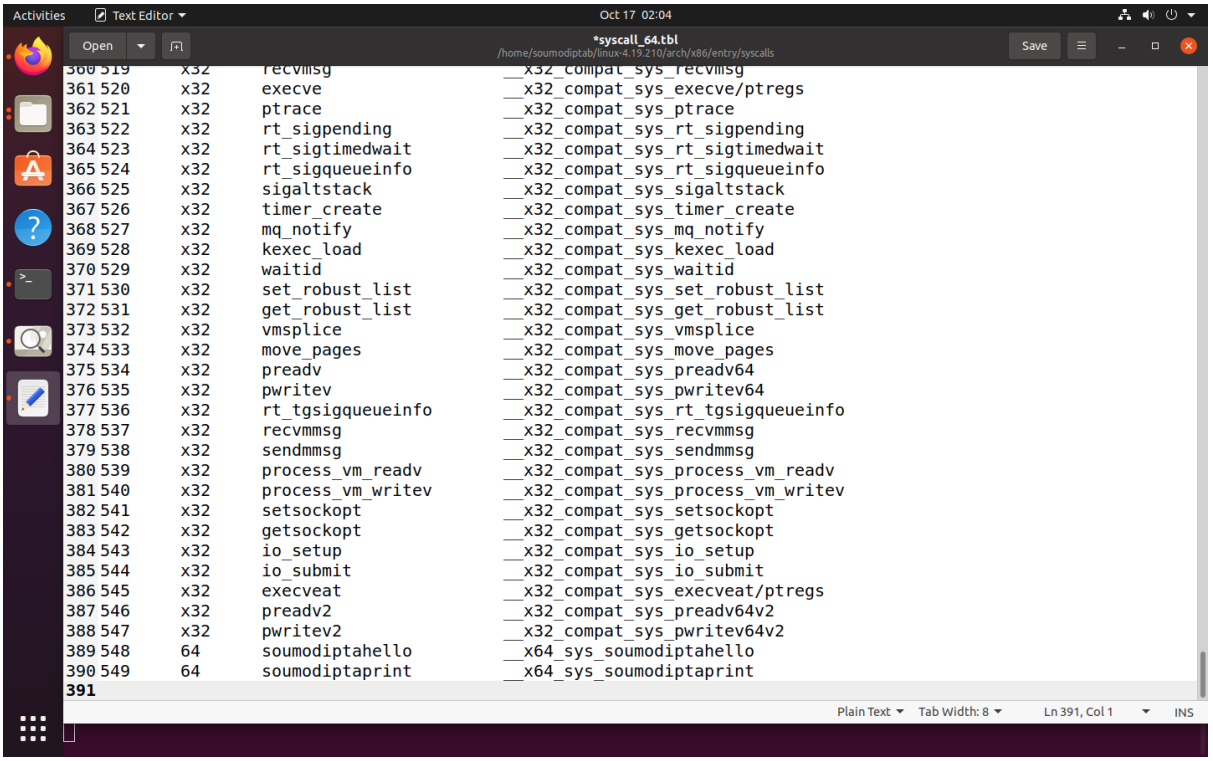
2. Traverse to **/custom/** and create a file using gedit: **soumodiptaprint.c**
3. Create a new system call called: **soumodiptaprint** with parameter of **char __user *.**



SYSCALL_DEFINE1 is a macro that will expand our system call using the internal scripts built inside the kernel and allow our system call to take one input parameter. Here the second and third argument are type of input and the variable name. Here I have used the **__user** macro so that we can refer to the address space of the user, without this it would have searched for the string inside the kernel space.

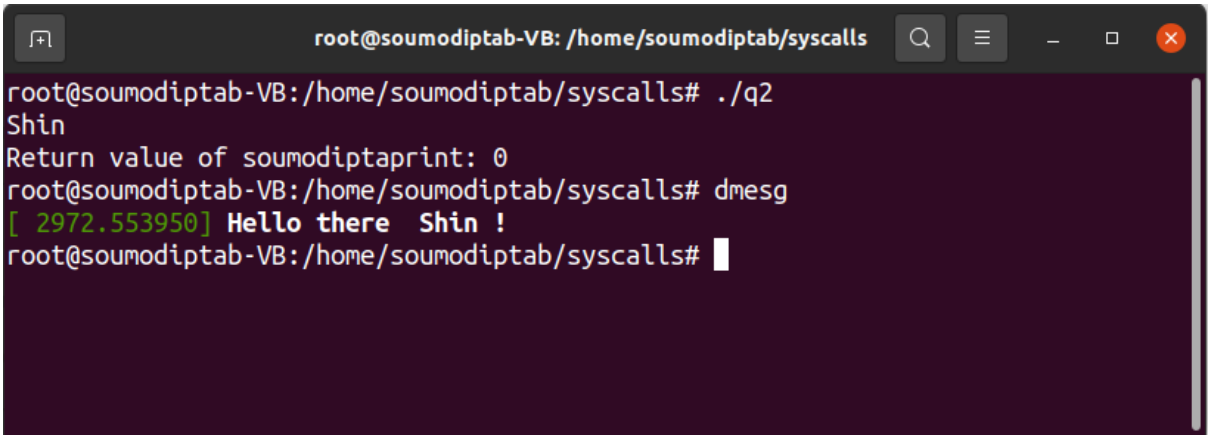
4. Add **soumodiptaprint.o** in the Makefile

5. Now traverse to **arch/x86/entry/syscalls/** and add a syscall entry in the table **syscall_64.tbl**



6. Traverse back to the root and use the **build** commands to compile the kernel and install them.
7. Reload the kernel from the **GRUB** menu.
8. Testing the system call using the following driver code.
9. For testing the system call I am using the following driver code.

10. Now checking in kernel logs using **dmesg**:



OBSERVATIONS:

- The memory is divided into two address spaces , one the kernel space and the other the user space. Without using the macro **__user** the address sent to our system call would have been searched only in the kernel address space, but using this keyword now the user space address will be referred. This is how we are able to print the string.

Question 3:

Write system call to print the parent process id and current process id upon calling it.

e.g., If your name is ayushi mahajan then syscall name will be ayushiprocess()

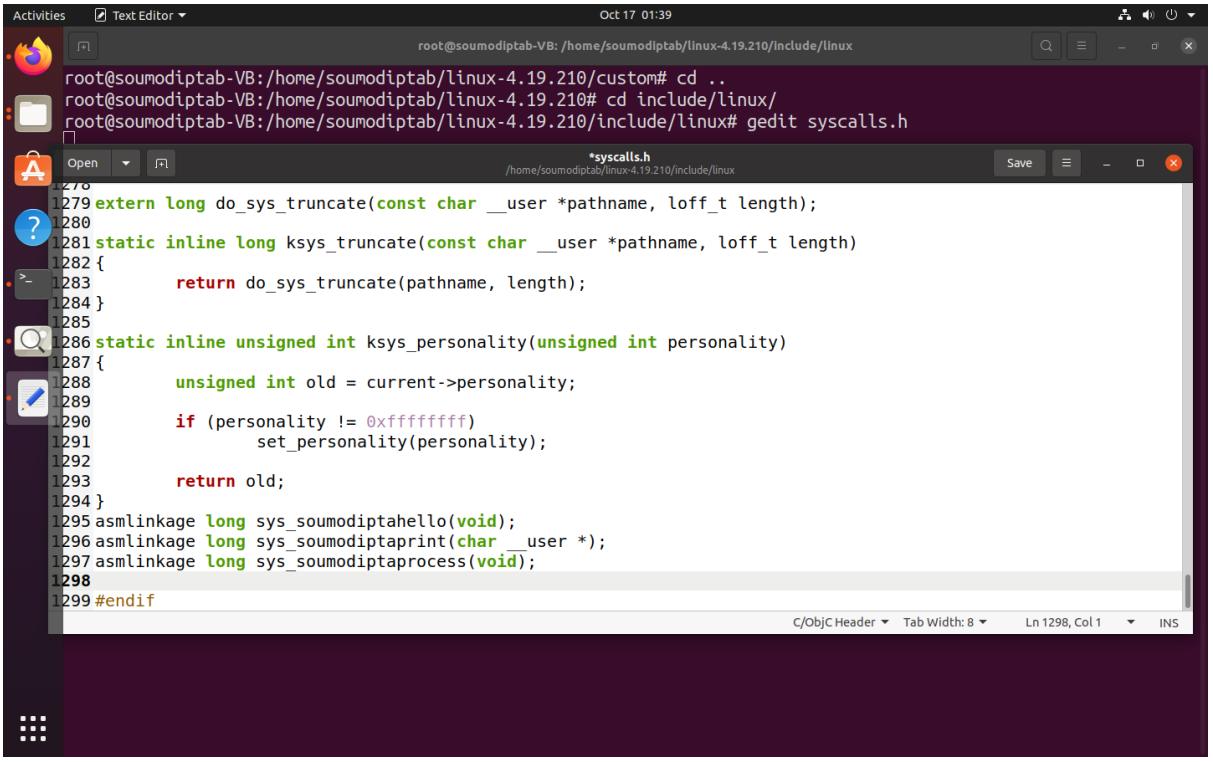
Which will print parent process id and current process id to Linux logs.

Are both process ids same or different? Why? What are your observations?

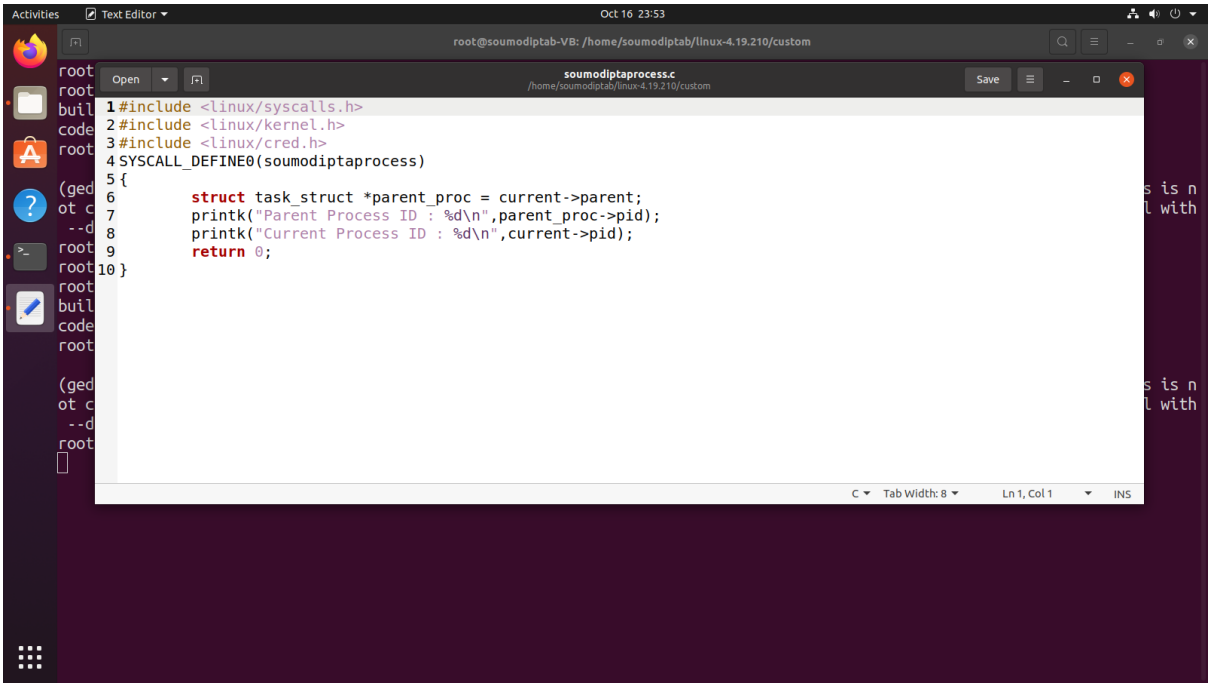
Solution:

Steps:

1. Traverse to **/include/linux/** and edit **syscalls.h** and add the definition of the system call.

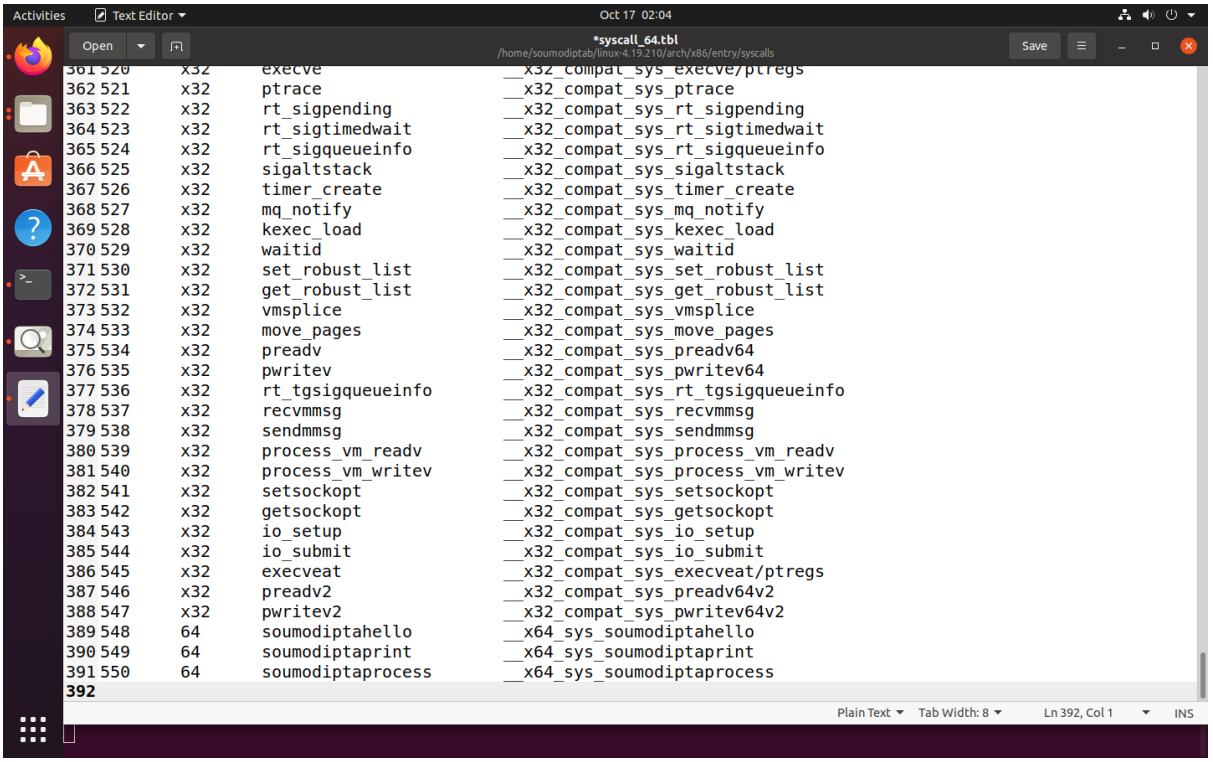


2. Traverse to **/custom/** and create a file using gedit: **soumodiptaprocess.c**
3. Create a new system call called **soumodiptaprocess**



Here I am using the process descriptor structure called **task_struct** which contains the process information, it is basically the process represented in memory. I am getting the reference of the **parent** process by using the **parent** pointer in the **current** process.Then I am extracting the **pid** attributes of both of them and logging them.A reference to **current** is present in **linux/cred.h**, so i am using this header to get the details of the current process.

4. Add **soumodiptaprint.o** in the Makefile
5. Now traverse to **arch/x86/entry/syscalls/** and add a syscall entry in the table **syscall_64.tbl**



6. Traverse back to the root and use the **build** commands to compile the kernel and install them.
7. Reload the kernel from the **GRUB** menu.

8. Testing the system call using the following driver code.
9. Now use the following driver code to test the system call:

Open q3.c ~/syscalls Save

```c
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>
int main()
{
        long int ret_value = syscall(550);
        printf("Syscall ret_value: %d\n",ret_value);
        return 0;
}
```

Saving file "/home/soumodiptab/syscalls/q3.c"... C Tab Width: 8 Ln 9, Col 9 INS

10. Now checking in **dmesg** for the output:

```
root@soumodiptab-VB: /home/soumodiptab/syscalls
root@soumodiptab-VB:/home/soumodiptab/syscalls# ./q3
Return value of soumodiptaprocess: 0
root@soumodiptab-VB:/home/soumodiptab/syscalls# dmesg
[13853.765145] Parent Process ID : 4708
[13853.765146] Current Process ID : 4844
root@soumodiptab-VB:/home/soumodiptab/syscalls# echo $$
4708
root@soumodiptab-VB:/home/soumodiptab/syscalls#
```

OBSERVATIONS:

- A process in nothing but a structure represented in memory as **task_struct** also known as process descriptor/task.
- **How do we know this is the correct parent process id ?**

For this I did a little experiment. First I examined the terminal process id and noted it down.

```
root@soumodiptab-VB:/home/soumodiptab/syscalls# echo $$
4708
```

Now if what I learned is correct then the terminal process is supposed to be the parent process of any process that gets started from here , that's true for when any object file that is run.

Now if we run our driver object code and look at the linux kernel logs then we will find :

```
root@soumodiptab-VB:/home/soumodiptab/syscalls# dmesg
[13853.765145] Parent Process ID : 4708
[13853.765146] Current Process ID : 4844
```

As we can see that the parent process id matches with the terminal process, so we can conclude that this is the correct parent process id that is being printed in the kernel logs.

- **Now about why the current and parent process id are different?**

When we run our driver code that is the object file, the terminal process issues a **fork()** system call that creates a replica of our original terminal process descriptor and then gives it a new process id. It also links the pointer called **parent** inside the structure **task_struct** with the parent process descriptor's address. So when we refer to **current->pid** and **parent_proc->pid** they are obviously different. When the driver code process terminates, the terminal's process descriptor is loaded again using the **parent** pointer.

Question 4:

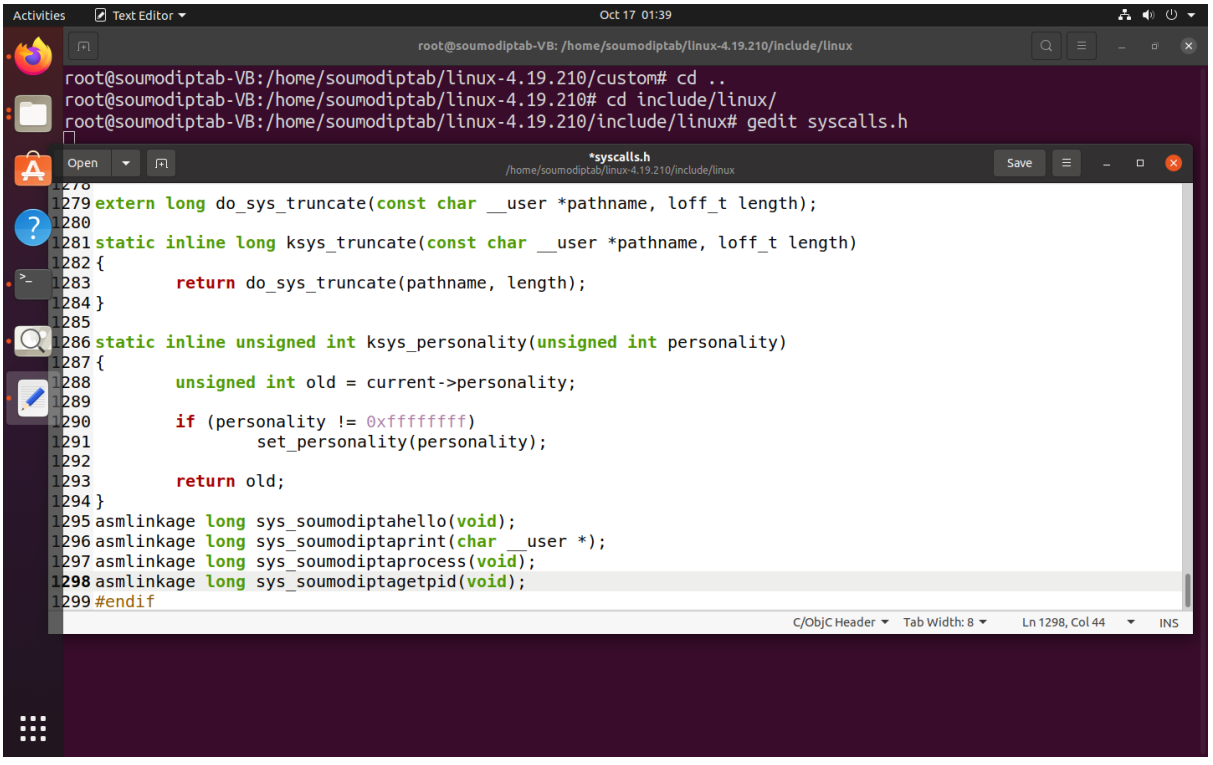
Write system call to execute some predefined system call from your written system call.

e.g., If your name is ayushi mahajan and you want to execute getpid() then syscall name will be
ayushigetpid()
Which will work the same way as getpid() works

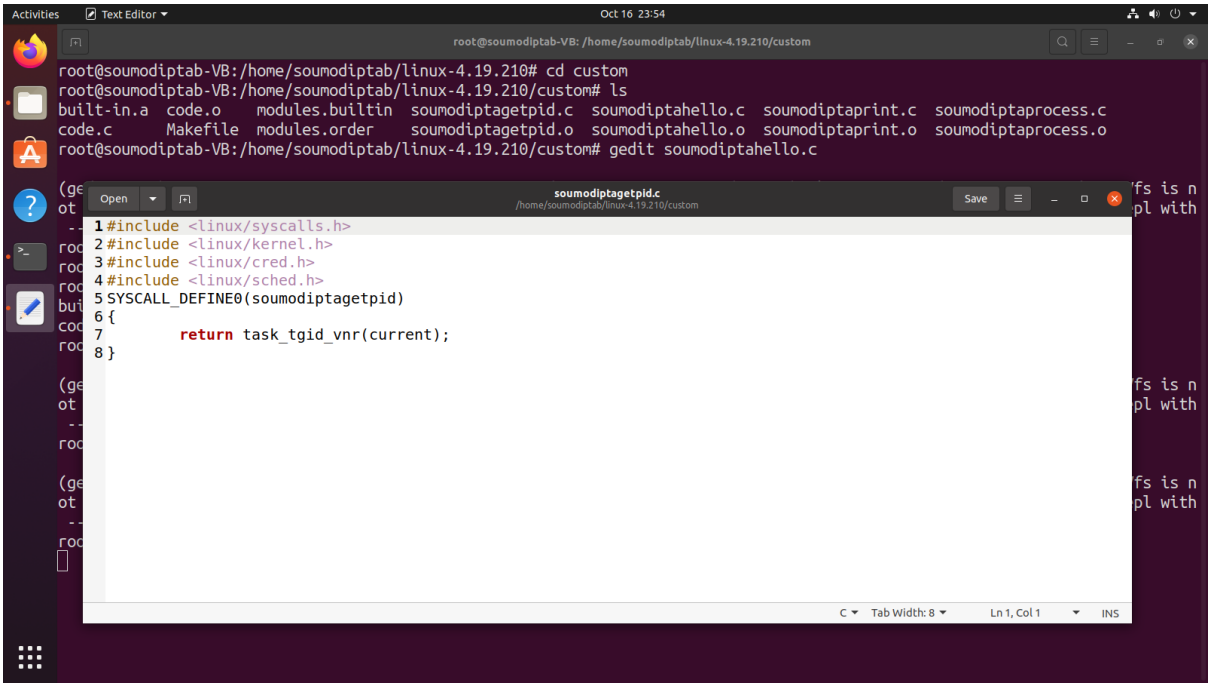
Solution:

Steps:

1. Traverse to **/include/linux/** and edit **syscalls.h** and add the definition of the system call.

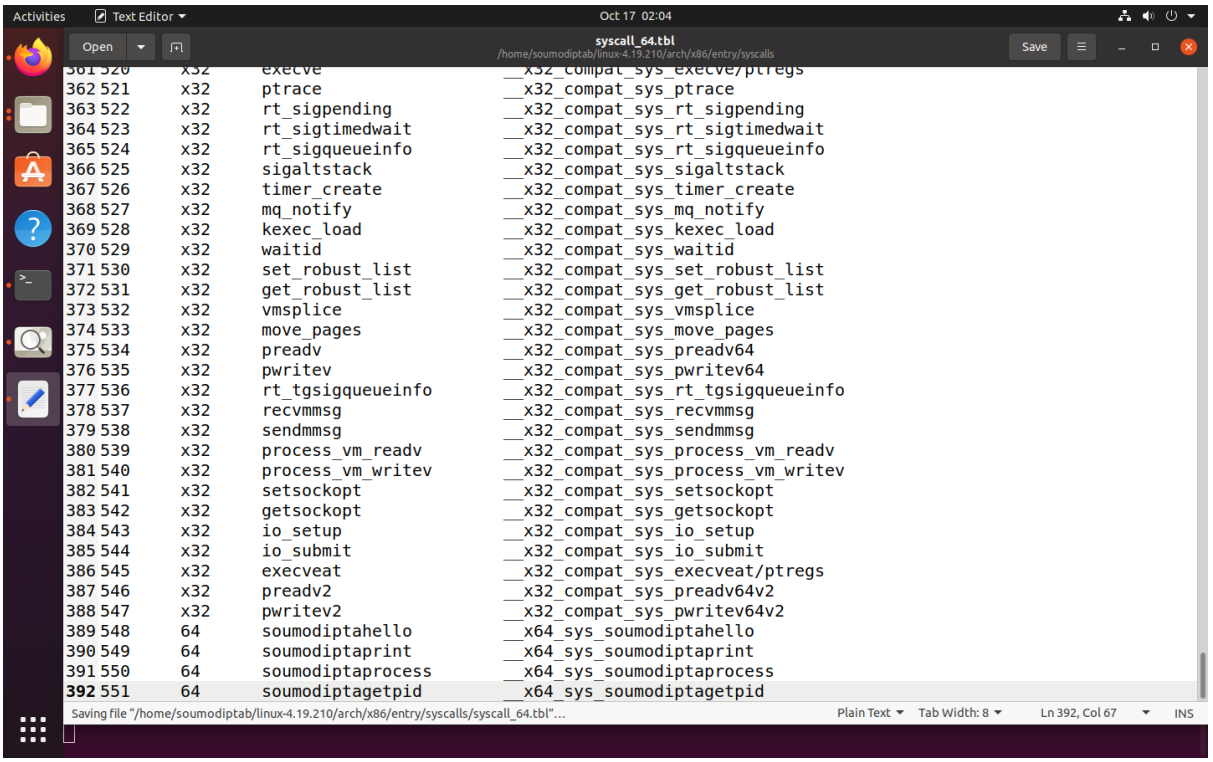


2. Traverse to **/custom/** and open file using gedit: **soumodiptagetpid.c**
3. Create a new system call called **soumodiptagetpid**



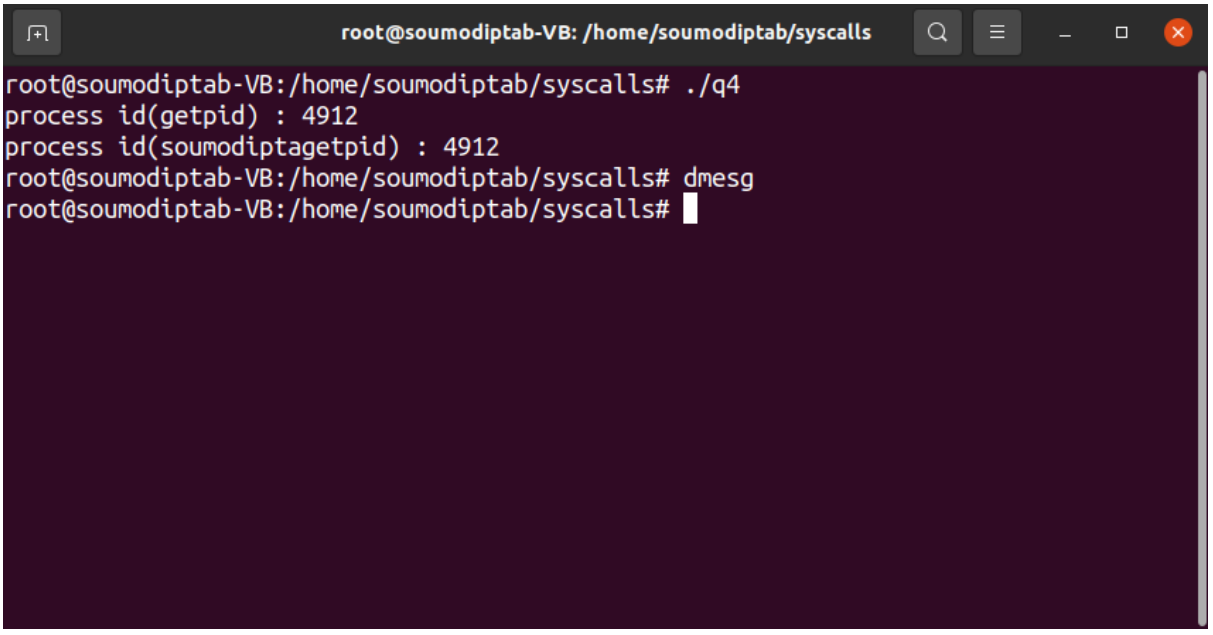
Note: I have used a kernel function here,there was no way to call a system call inside a kernel, so I proceeded with using a kernel function. **task_tgid_vnr()** is present in **linux/sched.h** which has **struct task_struct** as a parameter.
This function returns the process id of the task/process descriptor. In our case we are requesting the process id of the current task/process.

4. Add **soumodiptagetpid.o** in the Makefile
5. Now traverse to **arch/x86/entry/syscalls/** and add a syscall entry in the table **syscall_64.tbl**



6. Traverse back to the root and use the **build** commands to compile the kernel and install them.
7. Testing the system call using the following driver code.
8. Now we well test our system call using our driver code:

9. Now lets check the output:



OBSERVATIONS:

Both our custom syscall **soumodiptagetpid()** and original **getpid()** return the same Process id. This is because both the system calls refer to the same kernel function **task_tgid_vnr()**which returns the current process descriptor's **pid.**