

## 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.)

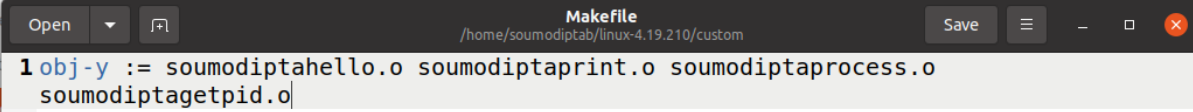
- **include/linux/** : **syscalls.h** ( where the system call needs to be declared )
- **arch/x86/entry/syscalls/** : **syscall\_64.tbl** ( contains the master syscalls table for my **x86\_64** system architecture )
- **custom/**: newly created folder where I have all my **c files** which contains the 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.



The screenshot shows a text editor window titled 'Makefile' with the path '/home/soumodiptab/linux-4.19.210/custom'. The content of the file is: `1 obj-y := soumodiptahello.o soumodiptaprint.o soumodiptaprocess.o soumodiptagetpid.o`

### 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:**

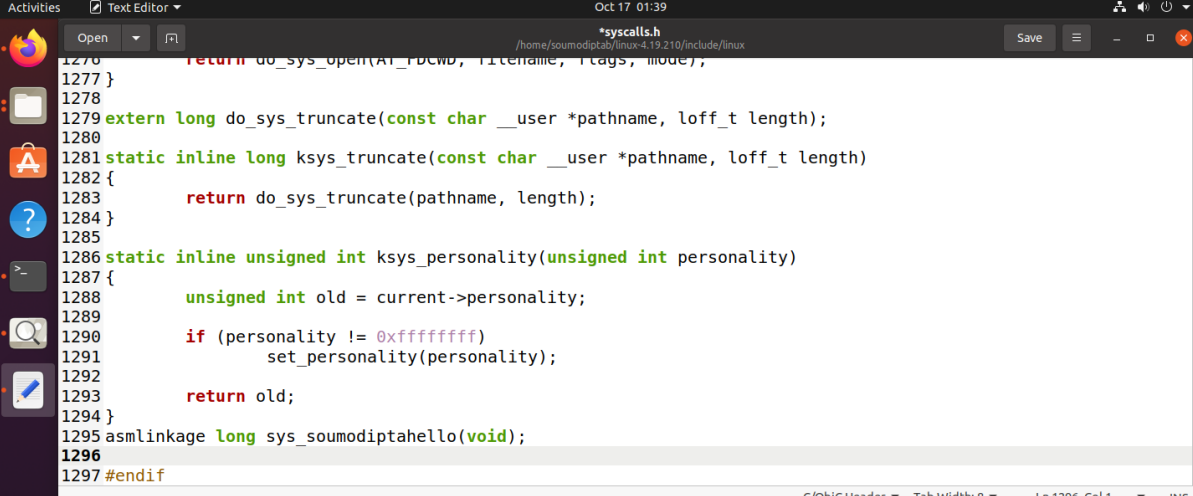
**Write syscall to print a welcome message to Linux logs.**

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.



```
1276 return do_sys_open(AT_FDCWD, filename, flags, mode),
1277 }
1278
1279 extern long do_sys_truncate(const char __user *pathname, loff_t length);
1280
1281 static inline long ksys_truncate(const char __user *pathname, loff_t length)
1282 {
1283     return do_sys_truncate(pathname, length);
1284 }
1285
1286 static inline unsigned int ksys_personality(unsigned int personality)
1287 {
1288     unsigned int old = current->personality;
1289     if (personality != 0xffffffff)
1290         set_personality(personality);
1291     return old;
1292 }
1293
1294 }
1295 asmlinkage long sys_soumodiptahello(void);
1296
1297 #endif
```

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



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

```
Open  q1.c
~/syscalls
1 #include <stdio.h>
2 #include <linux/kernel.h>
3 #include <sys/syscall.h>
4 #include <unistd.h>
5 int main()
6 {
7     long int ret_status = syscall(548);
8     printf("System call soumodiptahello returned %ld\n", ret_status);
9     return 0;
10 }
```

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.**

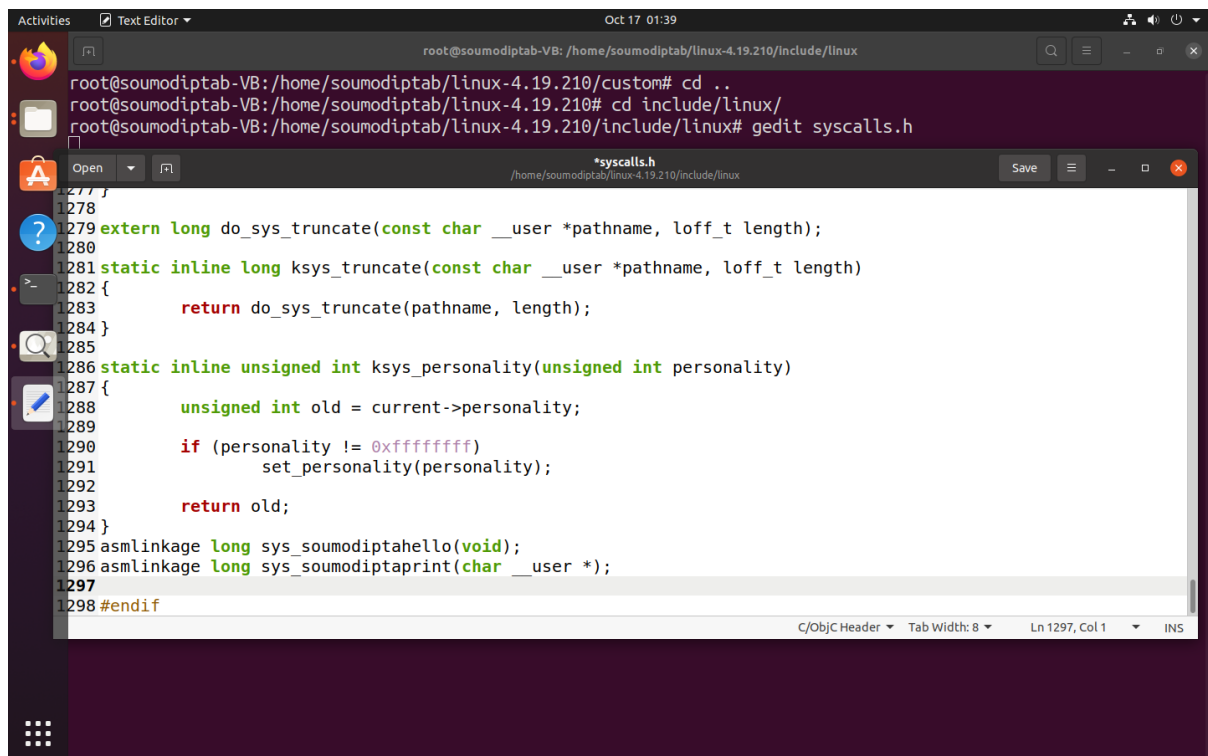
e.g., If your name is ayushi mahajan then syscall name will be `ayushiprint(string)`

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.

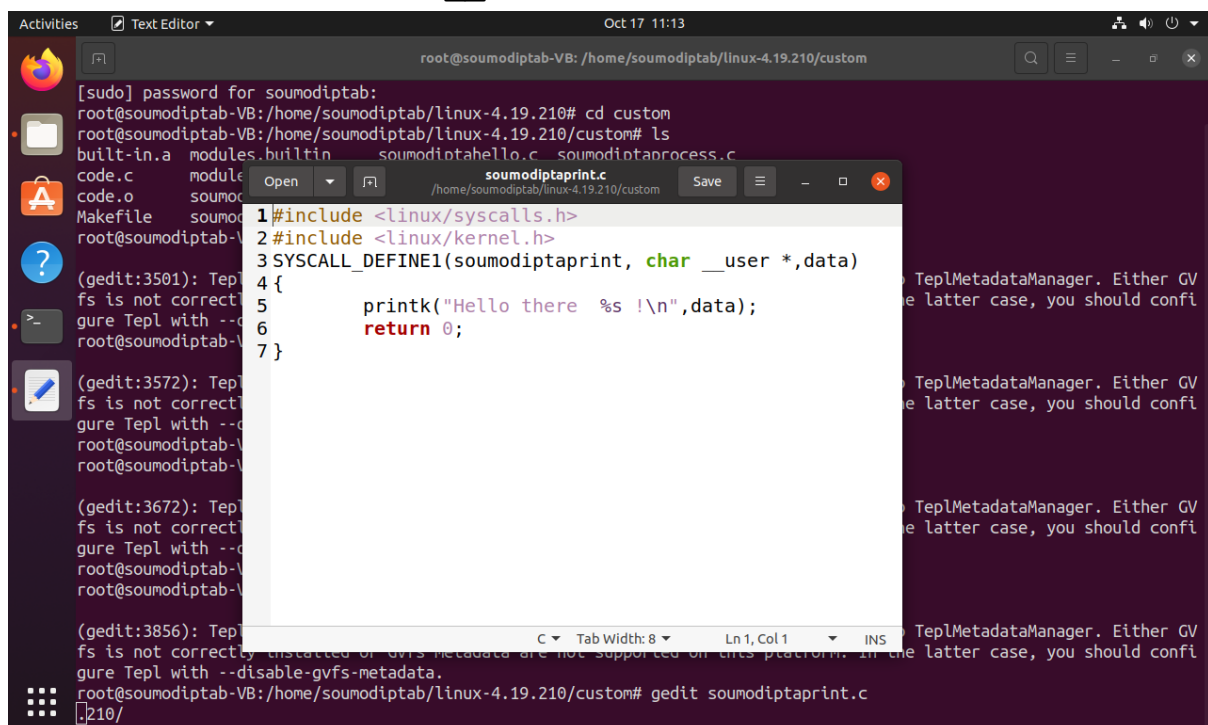


```
root@soumodiptab-VB: /home/soumodiptab/linux-4.19.210/include/linux
root@soumodiptab-VB: /home/soumodiptab/linux-4.19.210/custom# cd ..
root@soumodiptab-VB: /home/soumodiptab/linux-4.19.210# cd include/linux/
root@soumodiptab-VB: /home/soumodiptab/linux-4.19.210/include/linux# gedit syscalls.h

*syscalls.h
/home/soumodiptab/linux-4.19.210/include/linux

1277 }
1278
1279 extern long do_sys_truncate(const char __user *pathname, loff_t length);
1280
1281 static inline long ksys_truncate(const char __user *pathname, loff_t length)
1282 {
1283     return do_sys_truncate(pathname, length);
1284 }
1285
1286 static inline unsigned int ksys_personality(unsigned int personality)
1287 {
1288     unsigned int old = current->personality;
1289     if (personality != 0xffffffff)
1290         set_personality(personality);
1291     return old;
1292 }
1293
1294
1295 asmlinkage long sys_soumodiptahello(void);
1296 asmlinkage long sys_soumodiptaprint(char __user *);
1297
1298 #endif
```

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 \***.



```
[sudo] password for soumodiptab:
root@soumodiptab-VB: /home/soumodiptab/linux-4.19.210# cd custom
root@soumodiptab-VB: /home/soumodiptab/linux-4.19.210/custom# ls
built-in.a  modules.builtins  soumodiptahello.c  soumodiptaprocess.c
code.c      module            soumodiptaprint.c
code.o      soumod            soumodiptaprint.o
Makefile    soumod            soumodiptaprint.o
root@soumodiptab-VB: /home/soumodiptab/linux-4.19.210/custom# gedit soumodiptaprint.c

(gedit:3501): TeplMetad... Either GV
fs is not correctl... the latter case, you should confi
gure Tepl with --d...

(gedit:3572): TeplMetad... Either GV
fs is not correctl... the latter case, you should confi
gure Tepl with --d...

(gedit:3672): TeplMetad... Either GV
fs is not correctl... the latter case, you should confi
gure Tepl with --d...

(gedit:3856): TeplMetad... Either GV
fs is not correctly installed or GVfs Metadata are not supported on this platform. In the latter case, you should confi
gure Tepl with --disable-gvfs-metadata.
root@soumodiptab-VB: /home/soumodiptab/linux-4.19.210/custom# gedit soumodiptaprint.c

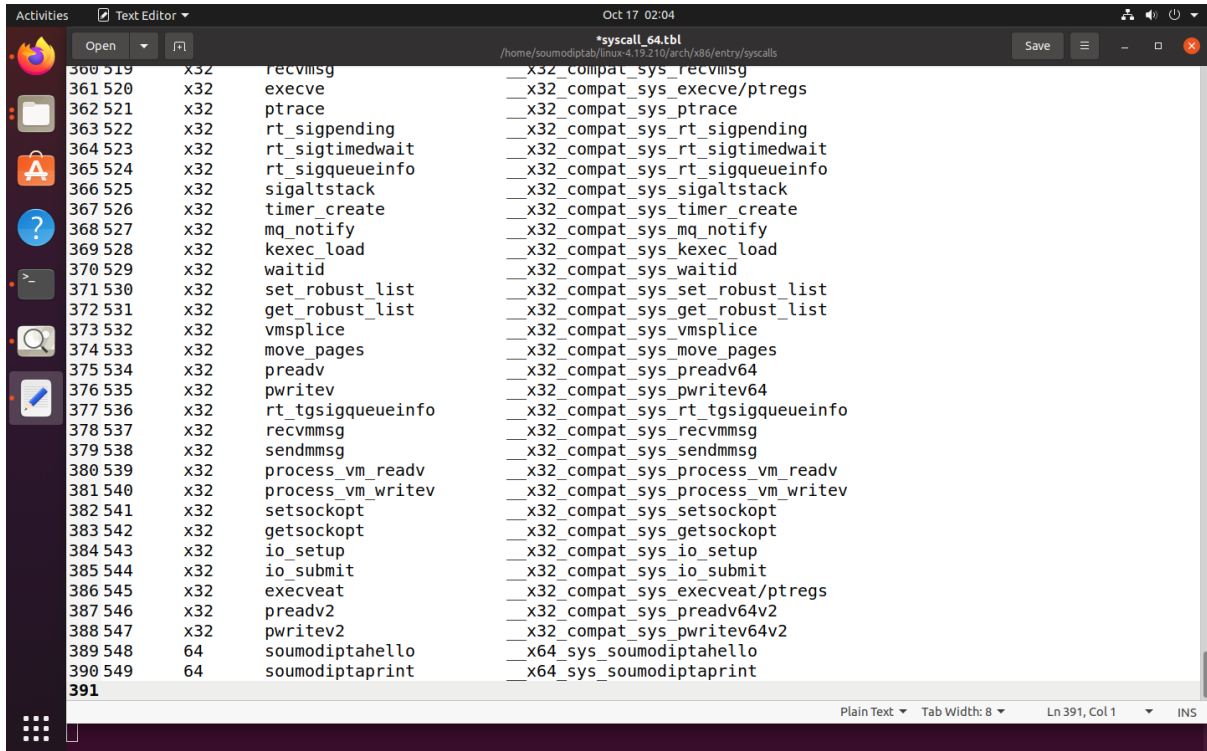
1#include <linux/syscalls.h>
2#include <linux/kernel.h>
3SYSCALL_DEFINE1(soumodiptaprint, char __user *,data)
4{
5    printk("Hello there  %s !\n",data);
6    return 0;
7}
```

**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.

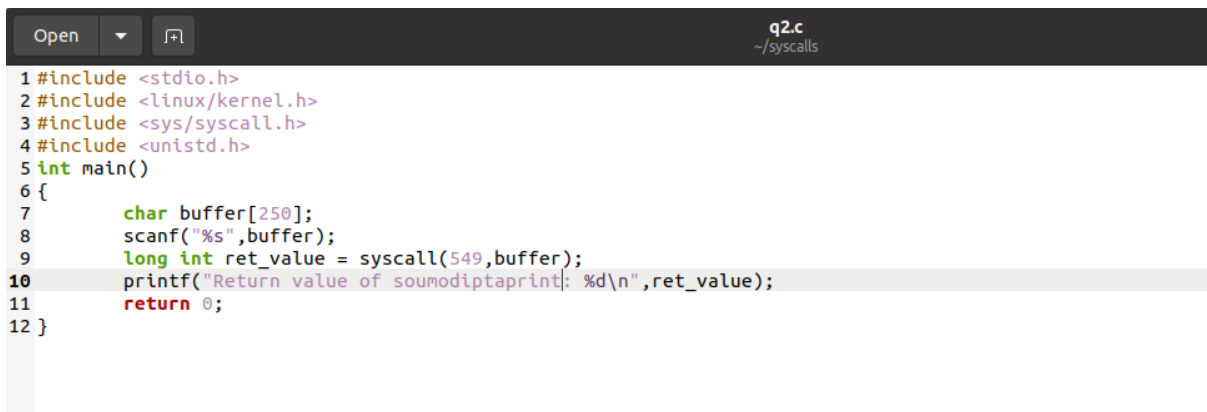
4. Add **soumodiptaprint.o** in the Makefile

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



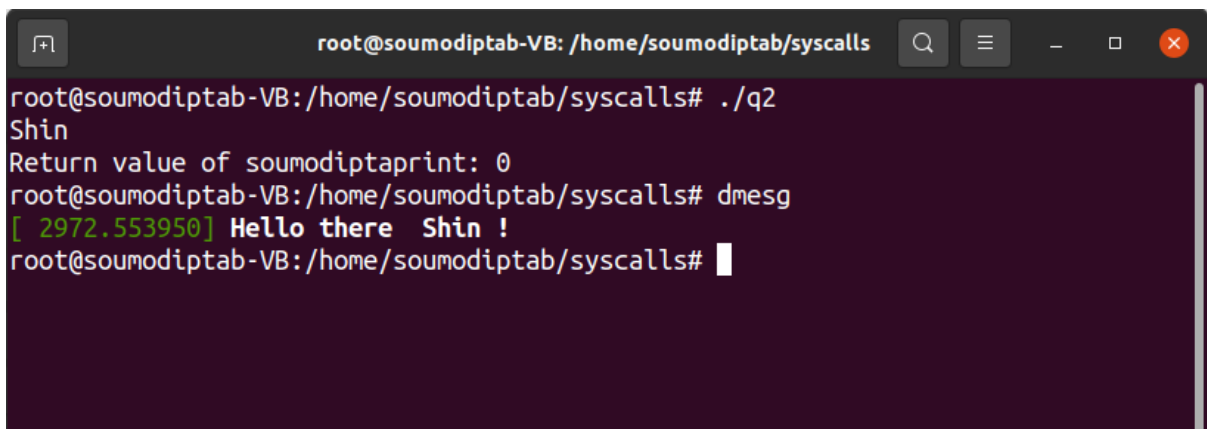
```
360 519 x32 recvmsg x32_compat_sys_recvmsg
361 520 x32 execve x32_compat_sys_execve/ptregs
362 521 x32 ptrace x32_compat_sys_ptrace
363 522 x32 rt_sigpending x32_compat_sys_rt_sigpending
364 523 x32 rt_sigtimedwait x32_compat_sys_rt_sigtimedwait
365 524 x32 rt_sigqueueinfo x32_compat_sys_rt_sigqueueinfo
366 525 x32 sigaltstack x32_compat_sys_sigaltstack
367 526 x32 timer_create x32_compat_sys_timer_create
368 527 x32 mq_notify x32_compat_sys_mq_notify
369 528 x32 kexec_load x32_compat_sys_kexec_load
370 529 x32 waitid x32_compat_sys_waitid
371 530 x32 set_robust_list x32_compat_sys_set_robust_list
372 531 x32 get_robust_list x32_compat_sys_get_robust_list
373 532 x32 vmsplice x32_compat_sys_vmsplice
374 533 x32 move_pages x32_compat_sys_move_pages
375 534 x32 preadv x32_compat_sys_preadv64
376 535 x32 pwritev x32_compat_sys_pwritev64
377 536 x32 rt_tgsigqueueinfo x32_compat_sys_rt_tgsigqueueinfo
378 537 x32 recvmmsg x32_compat_sys_recvmmsg
379 538 x32 sendmmsg x32_compat_sys_sendmmsg
380 539 x32 process_vm_readv x32_compat_sys_process_vm_readv
381 540 x32 process_vm_writev x32_compat_sys_process_vm_writev
382 541 x32 setsockopt x32_compat_sys_setsockopt
383 542 x32 getsockopt x32_compat_sys_getsockopt
384 543 x32 io_setup x32_compat_sys_io_setup
385 544 x32 io_submit x32_compat_sys_io_submit
386 545 x32 execveat x32_compat_sys_execveat/ptregs
387 546 x32 preadv2 x32_compat_sys_preadv64v2
388 547 x32 pwritev2 x32_compat_sys_pwritev64v2
389 548 64 soumodiptahello x64_sys_soumodiptahello
390 549 64 soumodiptaprint x64_sys_soumodiptaprint
391
```

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.



```
1 #include <stdio.h>
2 #include <linux/kernel.h>
3 #include <sys/syscall.h>
4 #include <unistd.h>
5 int main()
6 {
7     char buffer[250];
8     scanf("%s",buffer);
9     long int ret_value = syscall(549,buffer);
10    printf("Return value of soumodiptaprint: %d\n",ret_value);
11    return 0;
12 }
```

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



```
root@soumodiptab-VB: /home/soumodiptab/syscalls
root@soumodiptab-VB:/home/soumodiptab/syscalls# ./q2
Shin
Return value of soumodiptaprint: 0
root@soumodiptab-VB:/home/soumodiptab/syscalls# dmesg
[ 2972.553950] Hello there Shin !
root@soumodiptab-VB:/home/soumodiptab/syscalls#
```



## 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()`

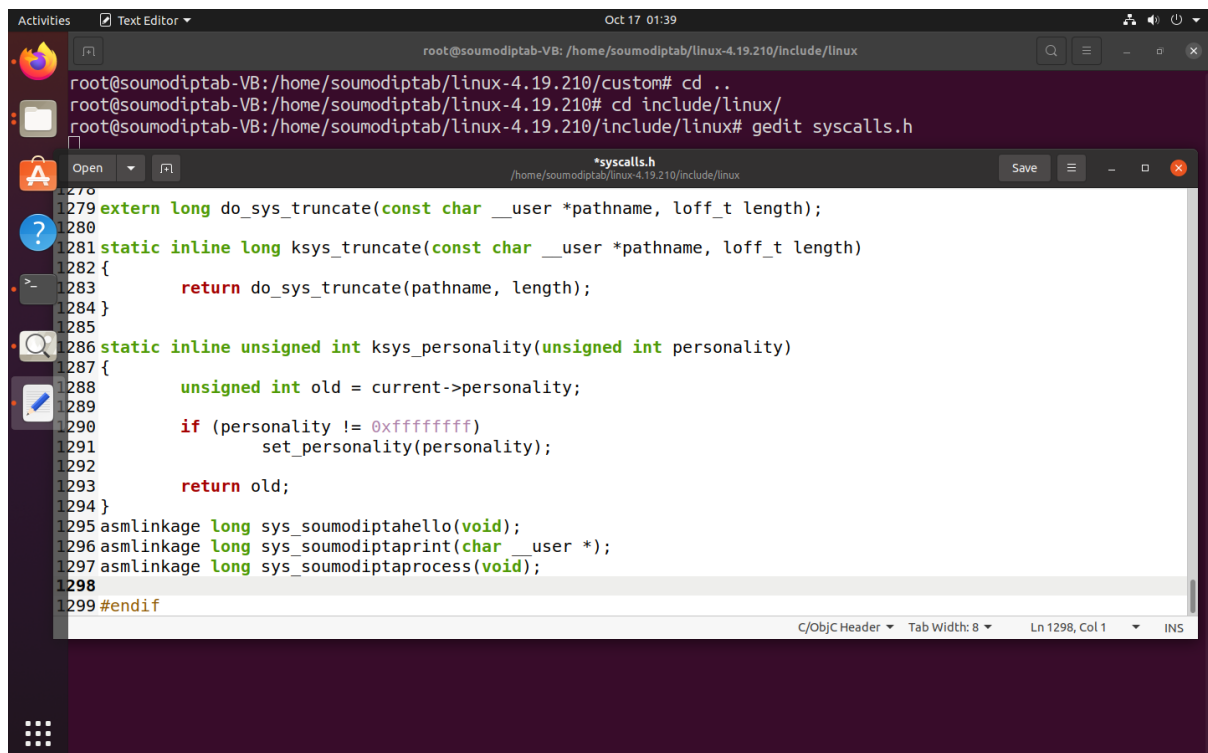
Which will print parent process id and current process id to Linux logs.

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.



The screenshot shows a terminal window and a text editor window. The terminal window displays the following commands and output:

```
root@soumodiptab-VB:/home/soumodiptab/linux-4.19.210/custom# cd ..
root@soumodiptab-VB:/home/soumodiptab/linux-4.19.210# cd include/linux/
root@soumodiptab-VB:/home/soumodiptab/linux-4.19.210/include/linux# gedit syscalls.h
```

The text editor window shows the `syscalls.h` file with the following code:

```
1279 extern long do_sys_truncate(const char __user *pathname, loff_t length);
1280
1281 static inline long ksys_truncate(const char __user *pathname, loff_t length)
1282 {
1283     return do_sys_truncate(pathname, length);
1284 }
1285
1286 static inline unsigned int ksys_personality(unsigned int personality)
1287 {
1288     unsigned int old = current->personality;
1289     if (personality != 0xffffffff)
1290         set_personality(personality);
1291     return old;
1292 }
1293
1294
1295 asmlinkage long sys_soumodiptahello(void);
1296 asmlinkage long sys_soumodiptaprint(char __user *);
1297 asmlinkage long sys_soumodiptaprocess(void);
1298
1299 #endif
```

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

```

1#include <linux/syscalls.h>
2#include <linux/kernel.h>
3#include <linux/cred.h>
4SYSCALL_DEFINE0(soumodiptaprocess)
5{
6    struct task_struct *parent_proc = current->parent;
7    printk("Parent Process ID : %d\n",parent_proc->pid);
8    printk("Current Process ID : %d\n",current->pid);
9    return 0;
10}

```

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**

361 520	x32	execve	x32_compat_sys_execve/ptregs
362 521	x32	ptrace	x32_compat_sys_ptrace
363 522	x32	rt_sigpending	x32_compat_sys_rt_sigpending
364 523	x32	rt_sigtimedwait	x32_compat_sys_rt_sigtimedwait
365 524	x32	rt_sigqueueinfo	x32_compat_sys_rt_sigqueueinfo
366 525	x32	sigaltstack	x32_compat_sys_sigaltstack
367 526	x32	timer_create	x32_compat_sys_timer_create
368 527	x32	mq_notify	x32_compat_sys_mq_notify
369 528	x32	kexec_load	x32_compat_sys_kexec_load
370 529	x32	waitid	x32_compat_sys_waitid
371 530	x32	set_robust_list	x32_compat_sys_set_robust_list
372 531	x32	get_robust_list	x32_compat_sys_get_robust_list
373 532	x32	vmsplice	x32_compat_sys_vmsplice
374 533	x32	move_pages	x32_compat_sys_move_pages
375 534	x32	preadv	x32_compat_sys_preadv64
376 535	x32	pwritev	x32_compat_sys_pwritev64
377 536	x32	rt_tgsigqueueinfo	x32_compat_sys_rt_tgsigqueueinfo
378 537	x32	recvmsg	x32_compat_sys_recvmsg
379 538	x32	sendmsg	x32_compat_sys_sendmsg
380 539	x32	process_vm_readv	x32_compat_sys_process_vm_readv
381 540	x32	process_vm_writev	x32_compat_sys_process_vm_writev
382 541	x32	setsockopt	x32_compat_sys_setsockopt
383 542	x32	getsockopt	x32_compat_sys_getsockopt
384 543	x32	io_setup	x32_compat_sys_io_setup
385 544	x32	io_submit	x32_compat_sys_io_submit
386 545	x32	execveat	x32_compat_sys_execveat/ptregs
387 546	x32	preadv2	x32_compat_sys_preadv64v2
388 547	x32	pwritev2	x32_compat_sys_pwritev64v2
389 548	64	soumodiptahello	x64_sys_soumodiptahello
390 549	64	soumodiptaprint	x64_sys_soumodiptaprint
391 550	64	soumodiptaprocess	x64_sys_soumodiptaprocess
392			

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:

```
q3.c
~/syscalls
Save
1 #include <stdio.h>
2 #include <linux/kernel.h>
3 #include <sys/syscall.h>
4 #include <unistd.h>
5 int main()
6 {
7     long int ret_value = syscall(550);
8     printf("Syscall ret_value: %d\n",ret_value);
9     return 0;
10 }
```

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 is 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.**

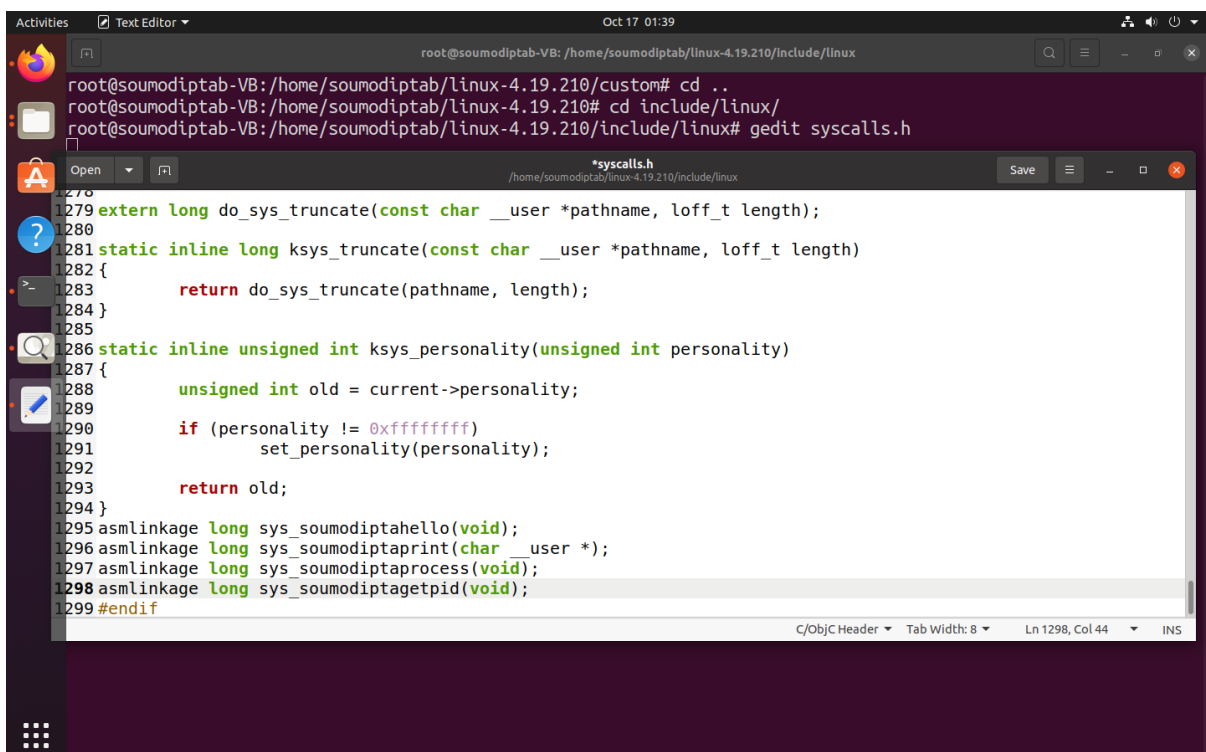
e.g., If your name is ayushi mahajan and you want to execute getpid() then syscall name will be ayushigetpid()

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.



The screenshot shows a terminal window and a text editor window. The terminal window displays the following commands and output:

```
root@soumodiptab-VB:/home/soumodiptab/linux-4.19.210/custom# cd ..
root@soumodiptab-VB:/home/soumodiptab/linux-4.19.210# cd include/linux/
root@soumodiptab-VB:/home/soumodiptab/linux-4.19.210/include/linux# gedit syscalls.h
```

The text editor window shows the **syscalls.h** file with the following code:

```
1278
1279 extern long do_sys_truncate(const char __user *pathname, loff_t length);
1280
1281 static inline long ksys_truncate(const char __user *pathname, loff_t length)
1282 {
1283     return do_sys_truncate(pathname, length);
1284 }
1285
1286 static inline unsigned int ksys_personality(unsigned int personality)
1287 {
1288     unsigned int old = current->personality;
1289     if (personality != 0xffffffff)
1290         set_personality(personality);
1291     return old;
1292 }
1293
1294
1295 asmlinkage long sys_soumodiptahello(void);
1296 asmlinkage long sys_soumodiptaprint(char __user *);
1297 asmlinkage long sys_soumodiptaprocess(void);
1298 asmlinkage long sys_soumodiptagetpid(void);
1299 #endif
```

The status bar at the bottom of the text editor shows "C/ObjC Header", "Tab Width: 8", "Ln 1298, Col 44", and "INS".

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

The screenshot shows a terminal window at the top and a gedit text editor window below it. The terminal shows the user navigating to the `/home/soumodiptab/linux-4.19.210/custom` directory and listing files. The gedit window shows the content of `soumodiptagetpid.c`.

```

root@soumodiptab-VB: /home/soumodiptab/linux-4.19.210/custom
root@soumodiptab-VB: /home/soumodiptab/linux-4.19.210/custom# ls
built-in.a code.o modules.builtins soumodiptagetpid.c soumodiptahello.c soumodiptaprint.c soumodiptaprocess.c
code.c Makefile modules.order soumodiptagetpid.o soumodiptahello.o soumodiptaprint.o soumodiptaprocess.o
root@soumodiptab-VB: /home/soumodiptab/linux-4.19.210/custom# gedit soumodiptahello.c

(gedit) soumodiptagetpid.c
/home/soumodiptab/linux-4.19.210/custom
1#include <linux/syscalls.h>
2#include <linux/kernel.h>
3#include <linux/cred.h>
4#include <linux/sched.h>
5SYSCALL_DEFINE0(soumodiptagetpid)
6{
7    return task_tgid_vnr(current);
8}
  
```

**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**

The screenshot shows a text editor window displaying the `syscall_64.tbl` file. The table lists system calls and their corresponding kernel functions. The new system call `soumodiptagetpid` has been added at the bottom of the table.

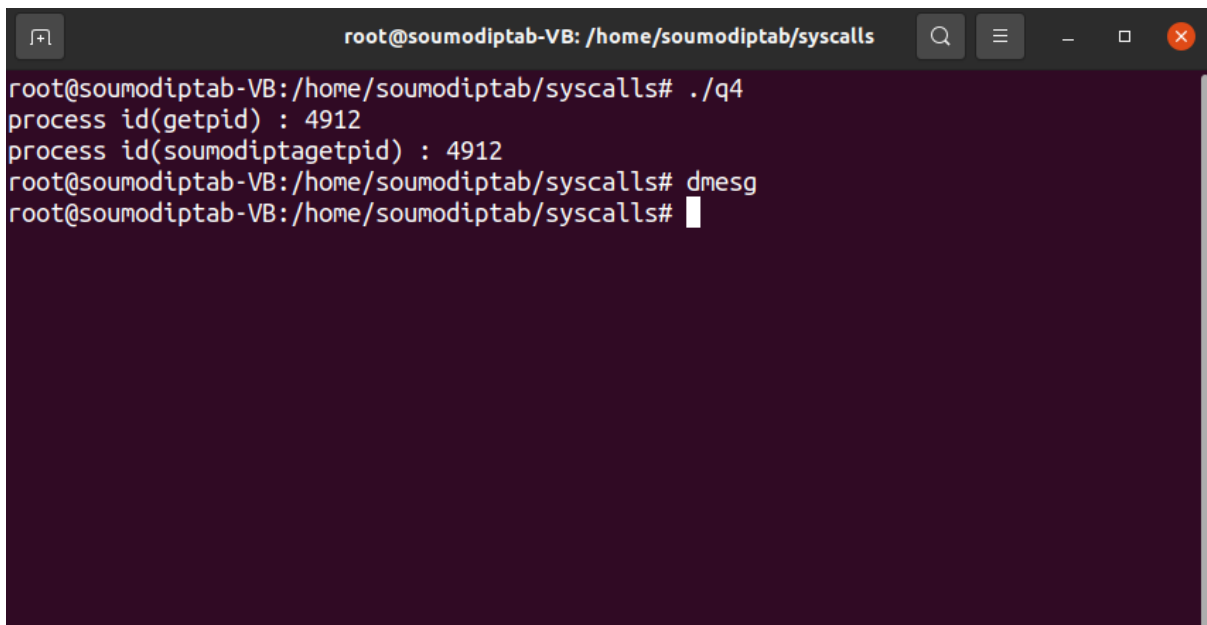
Number	Arch	Function	Kernel Function
361 520	x32	execve	x32_compat_sys_execve/ptregs
362 521	x32	ptrace	x32_compat_sys_ptrace
363 522	x32	rt_sigpending	x32_compat_sys_rt_sigpending
364 523	x32	rt_sigtimedwait	x32_compat_sys_rt_sigtimedwait
365 524	x32	rt_sigqueueinfo	x32_compat_sys_rt_sigqueueinfo
366 525	x32	sigaltstack	x32_compat_sys_sigaltstack
367 526	x32	timer_create	x32_compat_sys_timer_create
368 527	x32	mq_notify	x32_compat_sys_mq_notify
369 528	x32	kexec_load	x32_compat_sys_kexec_load
370 529	x32	waitid	x32_compat_sys_waitid
371 530	x32	set_robust_list	x32_compat_sys_set_robust_list
372 531	x32	get_robust_list	x32_compat_sys_get_robust_list
373 532	x32	vmsplce	x32_compat_sys_vmsplce
374 533	x32	move_pages	x32_compat_sys_move_pages
375 534	x32	preadv	x32_compat_sys_preadv64
376 535	x32	pwritev	x32_compat_sys_pwritev64
377 536	x32	rt_tgsigqueueinfo	x32_compat_sys_rt_tgsigqueueinfo
378 537	x32	recvmsg	x32_compat_sys_recvmsg
379 538	x32	sendmsg	x32_compat_sys_sendmsg
380 539	x32	process_vm_readv	x32_compat_sys_process_vm_readv
381 540	x32	process_vm_writev	x32_compat_sys_process_vm_writev
382 541	x32	setsockopt	x32_compat_sys_setsockopt
383 542	x32	getsockopt	x32_compat_sys_getsockopt
384 543	x32	io_setup	x32_compat_sys_io_setup
385 544	x32	io_submit	x32_compat_sys_io_submit
386 545	x32	execveat	x32_compat_sys_execveat/ptregs
387 546	x32	preadv2	x32_compat_sys_preadv64v2
388 547	x32	pwritev2	x32_compat_sys_pwritev64v2
389 548	64	soumodiptahello	x64_sys_soumodiptahello
390 549	64	soumodiptaprint	x64_sys_soumodiptaprint
391 550	64	soumodiptaprocess	x64_sys_soumodiptaprocess
392 551	64	soumodiptagetpid	x64_sys_soumodiptagetpid

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 will test our system call using our driver code:



```
1 #include <stdio.h>
2 #include <linux/kernel.h>
3 #include <sys/syscall.h>
4 #include <unistd.h>
5 int main()
6 {
7     pid_t id = syscall(551);
8     printf("process id(getpid) : %d\n", getpid());
9     printf("process id(soumodiptagetpid) : %d\n", id);
10    return 0;
11 }
```

9. Now let's check the output:



```
root@soumodiptab-VB: /home/soumodiptab/syscalls
root@soumodiptab-VB:/home/soumodiptab/syscalls# ./q4
process id(getpid) : 4912
process id(soumodiptagetpid) : 4912
root@soumodiptab-VB:/home/soumodiptab/syscalls# dmesg
root@soumodiptab-VB:/home/soumodiptab/syscalls#
```

#### 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**.