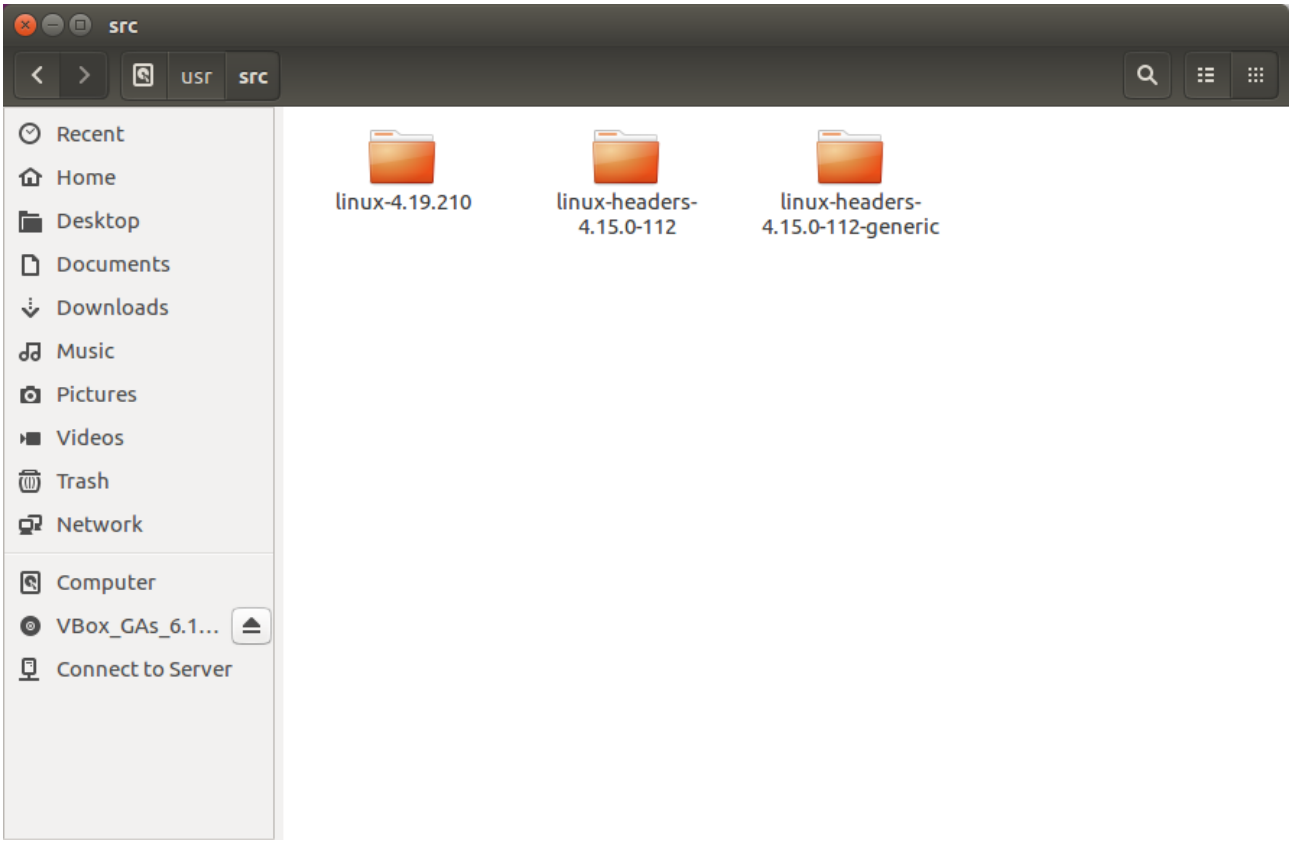
Advanced OS

Assignment #2 - Adding and testing a new system call to Linux kernel – Report

1. Setup

1. Downloaded Linux kernel version 4.19.210.
2. Extracted the kernel source code to /usr/src

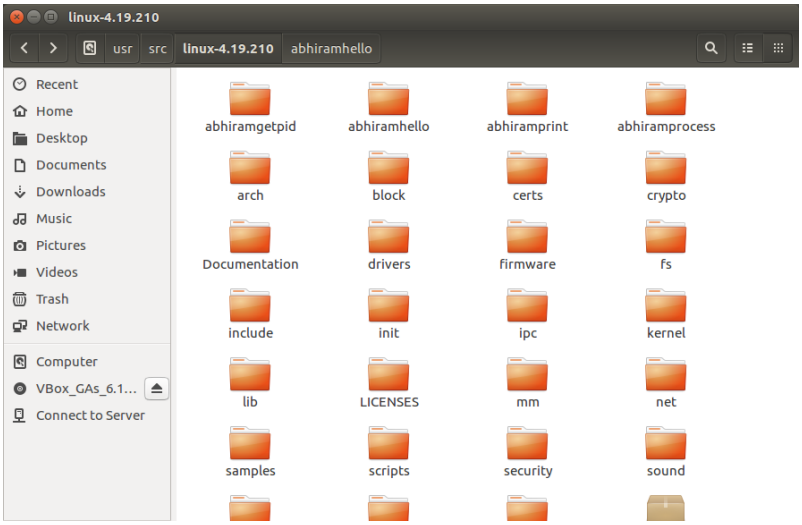
   `sudo tar -xvf` *linux-4.19.210.tar.gz -C/usr/src/*



Questions

inside */usr/src/linux-4.19.210.*
do *sudo -s* which allows the user to run the commands as root.

To make the 4 system calls in the question first 4 folders were made with the name of the system calls inside */usr/src/linux-4.19.210.*
(1)abhiramhello
(2)abhiramprint,
(3)abhiramprocess
(4)abhiramgetpid.



Question-1

In abhiramhello folder make a c file *abhiramhello.c* with code as shown below.

abhiramhello.c [Read-Only] (/usr/src/linux-4.19.210/abhiramhello) - gedit

Open Save

```c
#include <linux/kernel.h>

/* syscall-548 */

asmlinkage long abhiramhello(void){
        printk("Hello... Q1 seems to work fine. Welcome!!\n");
        return 0;
}
```

C Tab Width: 8 Ln 2, Col 1 INS

asmlinkage tag tells the compiler that the function should expect is arguments only from the CPU stack.

printk() is the function used by the kernel to print to the kernel log buffer. Kernel log buffer is a ring buffer which is exported to userspace through /dev/kmsg.

The function returns 0 only if its successfully completed.

Now create a text file named Makefile inside the abhiramhello folder as as shown below

Makefile [Read-Only] (/usr/src/linux-4.19.210/abhiramhello) - gedit

Open Save

```
obj-y := abhiramhello.o
```

Makefile Tab Width: 8 Ln 1, Col 24 INS

This ensures *abhiramhello.c* file is compiled and included in the kernel source code.

Question-2

Now go to abhiramprint folder in */usr/src/linux-4.19.210.* make a c file *abhiramprint.c* with code as shown below.

abhiramprint.c [Read-Only] (/usr/src/linux-4.19.210/abhiramprint) - gedit

Open Save

```c
#include <linux/kernel.h>
#include <linux/linkage.h>
#include <linux/syscalls.h>
#include <linux/sched.h>
#include <linux/cred.h>
#include <linux/uaccess.h>

/* prints input string alog with added message */
/* syscall number 549 */

SYSCALL_DEFINE1(abhiramprint, char __user *, source){
        char buf[256];

        if( copy_from_user(buf, source, sizeof(buf)) ){
                return -EFAULT;
        }

        printk("If the 2nd Question is working -> %s\n", buf);

        return 0;
}
```

C Tab Width: 8 Ln 9, Col 25 INS

SYSCALL_DEFINE1, here the 1 indicates one argument to the systemcall. Our system calls name is abhiramprint. Here the type of argument is char __user*(This denotes that this address is in userspace) and the name is source.

Now a buffer is declared which takes input from the source which carries the value that the user inputs and if any error occurs while copying from source to buffer we return -EFAULT(copy from user error).

After successfully copying the message its printed along with the given message.
Now create a text file named Makefile inside the abhiramprint folder as as shown below.
This ensures *abhiramprint.c* file is compiled and included in the kernel source code.

Makefile [Read-Only] (/usr/src/linux-4.19.210/abhiramprint) - gedit

Open Save

```makefile
obj-y := abhiramprint.o
```

Makefile Tab Width: 8 Ln 1, Col 1 INS

Question-3

1. Now go to abhiramprocess folder in */usr/src/linux-4.19.210.* make a c file *abhiramprocess.c* with code as shown below.

abhiramprocess.c [Read-Only] (/usr/src/linux-4.19.210/abhiramprocess) - gedit

Open Save

```c
#include <linux/kernel.h>
#include <linux/linkage.h>
#include <linux/syscalls.h>
#include <linux/sched.h>
#include <linux/cred.h>
#include <linux/uaccess.h>

asmlinkage long abhiramprocess(void){
        printk ("Current Process Id => %d\n", current->pid);
        printk ("Parent Process Id => %d\n", current->parent->pid);
        return 0;
}
```

C Tab Width: 8 Ln 1, Col 1 INS

Here current is a pointer to the current process. It points to the process that issued the system call. It is a global variable of *struct task_struct** . It comes from linux/sched.h.
So current->pid gives the process id of the current process, current->parent returns another *struct task_struct** variable which points to the parent process and current->parent->pid returns the process id of the parent process. We print these to the kernel logs using printk;

Now create a text file named Makefile inside the abhiramprocess folder as as shown below.

Makefile [Read-Only] (/usr/src/linux-4.19.210/abhiramprocess) - gedit

Open Save

abhiramprocess.c Makefile

```makefile
obj-y := abhiramprocess.o
```

Makefile Tab Width: 8 Ln 1, Col 1 INS

This ensures *abhiramprocess.c* file is compiled and included in the kernel source code.

Question-4

1. Now go to abhiramgetpid folder in */usr/src/linux-4.19.210.* make a c file *abhiramgetpid.c* with code as shown below.

abhiramgetpid.c [Read-Only] (/usr/src/linux-4.19.210/abhiramgetpid) - gedit

Open Save

abhiramprocess.c | Makefile | abhiramgetpid.c

```c
#include <linux/kernel.h>
#include <linux/linkage.h>
#include <linux/syscalls.h>
#include <linux/sched.h>
#include <linux/cred.h>
#include <linux/uaccess.h>

/* prints input string alog with added message */
/* syscall number 549 */

SYSCALL_DEFINE0(abhiramgetpid){
        return task_tgid_vnr(current);
}
```

C Tab Width: 8 Ln 1, Col 1 INS

getpid() system call, it is declared as

```c
SYSCALL_DEFINE0(getpid){
        return task_tgid_vnr(current);
}
```

Here we are using the same code for our systemcall so that our systemcall abhiramgetpid() also behaves the same as getpid(). It returns the current process id.

Now create a text file named Makefile inside the abhiramgetpid folder as as shown below. This ensures *abhiramgetpid.c* file is compiled and included in the kernel source code.

Makefile [Read-Only] (/usr/src/linux-4.19.210/abhiramgetpid) - gedit

Open Save

abhiramprocess.c | Makefile | abhiramgetpid.c | Makefile

```
obj-y := abhiramgetpid.o
```

Makefile Tab Width: 8 Ln 1, Col 25 INS

Adding our systemcalls to kernel's Makefile

Now goto */usr/src/linux-4.19.210* and Modify the kernels Makefile. Inside the Makefile search for the line that starts with core-y and on the second instance of the search we add our systems calls in the following manner.

Makefile [Read-Only] (/usr/src/linux-4.19.210) - gedit

```
    export SKIP_STACK_VALIDATION
  endif
endif

PHONY += prepare0

ifeq ($(KBUILD_EXTMOD),)
core-y          += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ abhiramhello/ abhiramprint/ abhiramprocess/ abhiramgetpid/

vmlinux-dirs    := $(patsubst %/,%,$(filter %/, $(init-y) $(init-m) \
                     $(core-y) $(core-m) $(drivers-y) $(drivers-m) \
                     $(net-y) $(net-m) $(libs-y) $(libs-m) $(virt-y)))

vmlinux-alldirs := $(sort $(vmlinux-dirs) $(patsubst %/,%,$(filter %/, \
                     $(init-) $(core-) $(drivers-) $(net-) $(libs-) $(virt-))))
```

Adding our systemcalls to system call table

Now goto */usr/src/linux-4.19.210/arch/x86/entry/syscalls/* and modify the syscall_64.tbl file as shown.

syscall_64.tbl [Read-Only] (/usr/src/linux-4.19.210/arch/x86/entry/syscalls) - gedit

```
522     x32     rt_sigpending           __x32_compat_sys_rt_sigpending
523     x32     rt_sigtimedwait         __x32_compat_sys_rt_sigtimedwait
524     x32     rt_sigqueueinfo         __x32_compat_sys_rt_sigqueueinfo
525     x32     sigaltstack             __x32_compat_sys_sigaltstack
526     x32     timer_create            __x32_compat_sys_timer_create
527     x32     mq_notify               __x32_compat_sys_mq_notify
528     x32     kexec_load              __x32_compat_sys_kexec_load
529     x32     waitid                  __x32_compat_sys_waitid
530     x32     set_robust_list         __x32_compat_sys_set_robust_list
531     x32     get_robust_list         __x32_compat_sys_get_robust_list
532     x32     vmsplice                __x32_compat_sys_vmsplice
533     x32     move_pages              __x32_compat_sys_move_pages
534     x32     preadv                  __x32_compat_sys_preadv64
535     x32     pwritev                 __x32_compat_sys_pwritev64
536     x32     rt_tgsigqueueinfo       __x32_compat_sys_rt_tgsigqueueinfo
537     x32     recvmmsg                __x32_compat_sys_recvmmsg
538     x32     sendmmsg                __x32_compat_sys_sendmmsg
539     x32     process_vm_readv        __x32_compat_sys_process_vm_readv
540     x32     process_vm_writev       __x32_compat_sys_process_vm_writev
541     x32     setsockopt              __x32_compat_sys_setsockopt
542     x32     getsockopt              __x32_compat_sys_getsockopt
543     x32     io_setup                __x32_compat_sys_io_setup
544     x32     io_submit               __x32_compat_sys_io_submit
545     x32     execveat                __x32_compat_sys_execveat/ptregs
546     x32     preadv2                 __x32_compat_sys_preadv64v2
547     x32     pwritev2                __x32_compat_sys_pwritev64v2
548     64      abhiramhello            abhiramhello
549     64      abhiramprocess          abhiramprocess
```

Here I have added abhiramhello and abhiramprocess as 548 and 549 systemcalls. These numbers should not already exist as these numbers uniquely identify as system call. The second column says for 64bit systems. Third column is the systemcall name and the fourth column is the function name.

syscall_64.tbl [Read-Only] (/usr/src/linux-4.19.210/arch/x86/entry/syscalls) - gedit

Open Save

abhiramprocess.c × Makefile × abhiramgetpid.c × Makefile × Makefile × syscall_64.tbl ×

```
325     common  mlock2                  __x64_sys_mlock2
326     common  copy_file_range         __x64_sys_copy_file_range
327     64      preadv2                 __x64_sys_preadv2
328     64      pwritev2                __x64_sys_pwritev2
329     common  pkey_mprotect           __x64_sys_pkey_mprotect
330     common  pkey_alloc              __x64_sys_pkey_alloc
331     common  pkey_free               __x64_sys_pkey_free
332     common  statx                   __x64_sys_statx
333     common  io_pgetevents           __x64_sys_io_pgetevents
334     common  rseq                    __x64_sys_rseq
335     common  abhiramprint            __x64_sys_abhiramprint
336     common  abhiramgetpid           __x64_sys_abhiramgetpid

#
# x32-specific system call numbers start at 512 to avoid cache impact
# for native 64-bit operation. The __x32_compat_sys stubs are created
# on-the-fly for compat_sys_*() compatibility system calls if X86_X32
# is defined.
#
512     x32     rt_sigaction            __x32_compat_sys_rt_sigaction
513     x32     rt_sigreturn            sys32_x32_rt_sigreturn
514     x32     ioctl                   __x32_compat_sys_ioctl
515     x32     readv                   __x32_compat_sys_readv
516     x32     writev                  __x32_compat_sys_writev
517     x32     recvfrom                __x32_compat_sys_recvfrom
518     x32     sendmsg                 __x32_compat_sys_sendmsg
519     x32     recvmsg                 __x32_compat_sys_recvmsg
520     x32     execve                  __x32_compat_sys_execve/ptregs
```

Plain Text Tab Width: 8 Ln 389, Col 45 INS

Here I have added abhiramprint and abhiramgetpid as the 335 and 336 systemcalls. The second column common says, it common for all systems. Here as you can see thewe used SYSCALL_DEFINEx macro for these systemcalls which adds the prefix of __x64_sys_ to our function name. So we specify the modified name in the fourth column.

Adding our new systemcalls to the system call header file

Now goto */usr/src/linux-4.19.210/include/linux/* and modify the syscalls.h file as shown.

syscalls.h [Read-Only] (/usr/src/linux-4.19.210/include/linux) - gedit

Open Save

Makefile × abhiramgetpid.c × Makefile × Makefile × syscall_64.tbl × syscalls.h ×

```
{
        if (force_o_largefile())
                flags |= O_LARGEFILE;
        return do_sys_open(AT_FDCWD, filename, flags, mode);
}

extern long do_sys_truncate(const char __user *pathname, loff_t length);

static inline long ksys_truncate(const char __user *pathname, loff_t length)
{
        return do_sys_truncate(pathname, length);
}

static inline unsigned int ksys_personality(unsigned int personality)
{
        unsigned int old = current->personality;

        if (personality != 0xffffffff)
                set_personality(personality);

        return old;
}
asmlinkage long abhiramhello(void);
asmlinkage long abhiramprint(char __user*);
asmlinkage long abhiramprocess(void);
asmlinkage long abhiramgetpid(void);

#endif
```

C/C++/ObjC Header Tab Width: 8 Ln 1287, Col 2 INS

Here we define our systemcall prototypes to the end of the file just before #endif statement.“asmlinkage” is a key word used to indicate that all parameters of the function would be available on the stack.

Adding our new systemcalls to the system call header file

Type the following commands in the terminal.

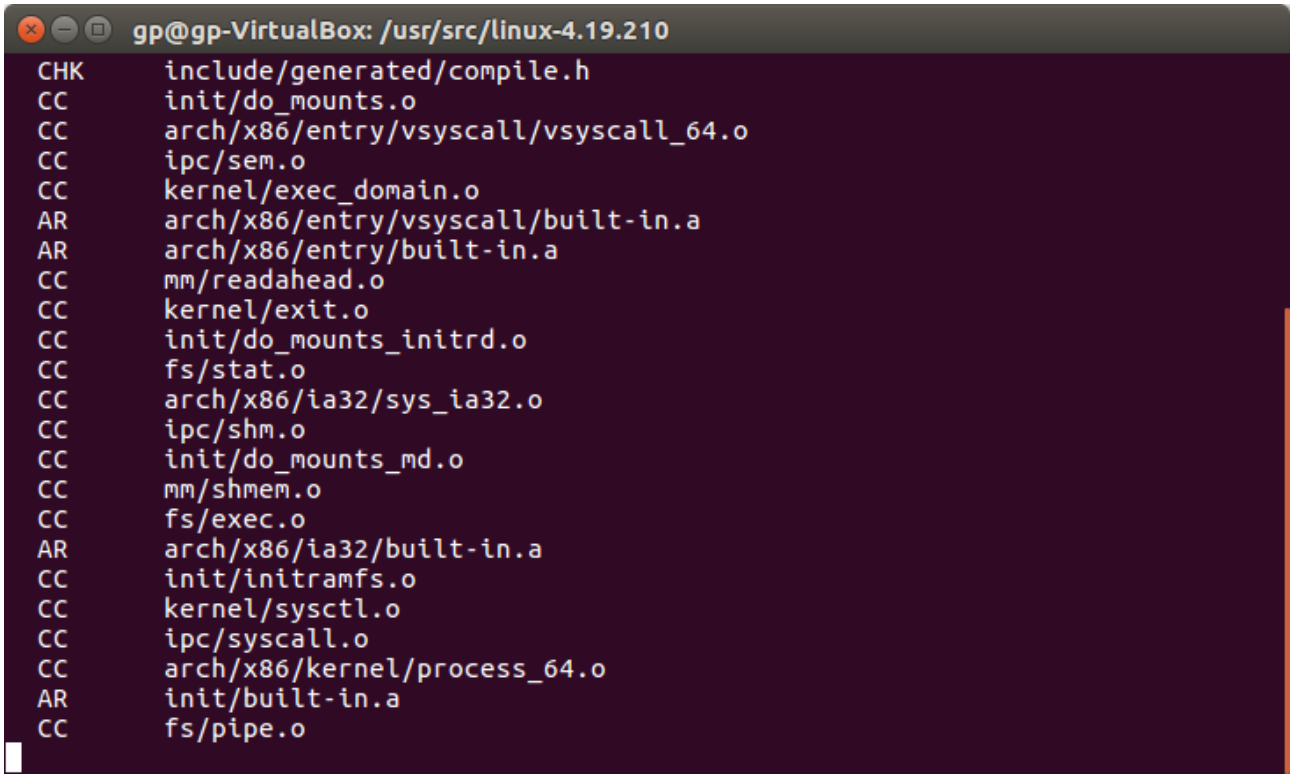
```
sudo apt-get install gcc
sudo apt-get install libncurses5-dev
sudo apt-get install bison
sudo apt-get install flex
sudo apt-get install libssl-dev
sudo apt-get install libelf-dev
sudo apt-get update
sudo apt-get upgrade
```

These installs all the necessary dependencies for us to compile the kernel successfully.

Now to configure our kernel go to */usr/src/linux-4.19.210* and execute the command `sudo make menuconfig`

A popup will come where we should navigate to File-Systems menu and select ext4 option and save the configuration.

Now to compile the kernel use *sudo make -j6*
This will use 6 cores to compile the kerenel which makes the compilation run faster. Then compilation begins and it may take sometime. For me it took within 40 minutes



Install or Update the Kernel

Then run the command `sudo make modules_install install` in the terminal. It will create some files in */boot/* directory and will make entries in grub.cfg.
Now check if the following fies are present in the boot directory.

1. System.map-4.19.210
2. vmlinuz-4.19.210
3. initrd.img-4.19.210
4. config-4.19.210

```
root@gp-VirtualBox: /boot
ls-lah: command not found
gp@gp-VirtualBox:/boot$ sudo -s
root@gp-VirtualBox:/boot# ls -lah
total 1.1G
drwxr-xr-x  3 root root 4.0K Sep 15 07:50 .
drwxr-xr-x 24 root root 4.0K Sep 15 06:13 ..
-rw-r--r--  1 root root 213K Jul 10  2020 config-4.15.0-112-generic
-rw-r--r--  1 root root 213K Apr 13  2021 config-4.15.0-142-generic
-rw-r--r--  1 root root 215K Sep 15 07:49 config-4.19.210
-rw-r--r--  1 root root 215K Sep 14 17:27 config-4.19.210.old
drwxr-xr-x  5 root root 4.0K Sep 15 07:50 grub
-rw-r--r--  1 root root  54M Sep 14 11:07 initrd.img-4.15.0-112-generic
-rw-r--r--  1 root root  54M Sep 15 06:13 initrd.img-4.15.0-142-generic
-rw-r--r--  1 root root 458M Sep 15 07:50 initrd.img-4.19.210
-rw-r--r--  1 root root 458M Sep 14 12:01 initrd.img-4.19.210.dpkg-bak
-rw-r--r--  1 root root 179K Jan 28  2016 memtest86+.bin
-rw-r--r--  1 root root 181K Jan 28  2016 memtest86+.elf
-rw-r--r--  1 root root 181K Jan 28  2016 memtest86+_multiboot.bin
-rw-------  1 root root 3.9M Jul 10  2020 System.map-4.15.0-112-generic
-rw-------  1 root root 3.9M Apr 13  2021 System.map-4.15.0-142-generic
-rw-r--r--  1 root root 4.1M Sep 15 07:49 System.map-4.19.210
-rw-r--r--  1 root root 4.1M Sep 14 17:27 System.map-4.19.210.old
-rw-r--r--  1 root root 7.9M Aug  7  2020 vmlinuz-4.15.0-112-generic
-rw-------  1 root root 7.9M Apr 15  2021 vmlinuz-4.15.0-142-generic
-rw-r--r--  1 root root 8.2M Sep 15 07:49 vmlinuz-4.19.210
-rw-r--r--  1 root root 8.2M Sep 14 17:27 vmlinuz-4.19.210.old
root@gp-VirtualBox:/boot#
```

Now reboot the system. After rebooting lets verify the kernel version using *uname -r*

```
gp@gp-VirtualBox: ~
gp@gp-VirtualBox:~$ uname -r
4.19.210
gp@gp-VirtualBox:~$
```

It will show our newly compiled version of 4.19.210

Testing our system calls

Go to the home(~) directory and make a file named userspace.c where we will write c code to call our new systemcalls. Write the following code.

```
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>
int main(){
        long int returnVal = syscall(548);
        printf("System call abhiramhello returned: %ld\n", returnVal);

        char st[256];
        sprintf(st, "lets do Question 3");
        long int stat = syscall(335, st);
        printf("System call abhiramprint returned: %ld\n", stat);

        stat = syscall(549);
        printf("System call abhiramprocess returned: %ld\n", stat);

        returnVal = syscall(336);
        if(returnVal>=0) printf("System call abhiramgetpid returned the current
process id => %ld\n", returnVal);
        else printf("System call returning negative value (error)!");
        return 0;
}
```

syscall(system_call_number) is used to invoke our system calls.
First we call syscall(548) which is the abhiramhello system call. It should return 0 and print the message in the kernel logs.

Then we call abhiram print using syscall(335, st) where st is the argument with the string that we are providing. It should return 0 on success and print the modified string in the kernel logs.

Then we call syscall(549) which is abhiram process and it should return 0 if successfull and print the child and parent process ids in the kernel logs.

Then finally we call syscall(336) which will return the process id of current process(behaves the same as getpid system call) and should it fail, it will return a negative value which will be printed in the logs as an error message.

Now compile this c file using *gcc userspace.c* and run it using *./a.out.* We will get the following output.

```
gp@gp-VirtualBox: ~
gp@gp-VirtualBox:~$ gcc userspace.c
gp@gp-VirtualBox:~$ ./a.out
System call abhiramhello returned: 0
System call abhiramprint returned: 0
System call abhiramprocess returned: 0
System call abhiramgetpid returned the current process id => 2811
gp@gp-VirtualBox:~$
```

Now lets check the ernel logs for our messages using *dmesg*

```
gp@gp-VirtualBox: ~
[    8.708189] vboxvideo: loading version 6.1.38 r153438
[    8.926995] 07:12:30.041071 main     VBoxService 6.1.38 r153438 (verbosity: 0
) linux.amd64 (Sep  1 2022 15:42:08) release log
               07:12:30.041073 main     Log opened 2022-09-15T07:12:30.041065000
Z
[    8.927028] 07:12:30.041134 main     OS Product: Linux
[    8.927049] 07:12:30.041158 main     OS Release: 4.19.210
[    8.927069] 07:12:30.041179 main     OS Version: #4 SMP Thu Sep 15 07:43:15 I
ST 2022
[    8.927096] 07:12:30.041199 main     Executable: /opt/VBoxGuestAdditions-6.1.
38/sbin/VBoxService
               07:12:30.041200 main     Process ID: 1181
               07:12:30.041200 main     Package type: LINUX_64BITS_GENERIC
[    8.929009] 07:12:30.043111 main     6.1.38 r153438 started. Verbose level =
0
[    8.930062] 07:12:30.044141 main     vbglR3GuestCtrlDetectPeekGetCancelSuppor
t: Supported (#1)
[   14.801648] ISO 9660 Extensions: Microsoft Joliet Level 3
[   14.802422] ISO 9660 Extensions: RRIP_1991A
[ 1529.836452] Hello... Q1 seems to work fine. Welcome!!
[ 1529.836506] If the 2nd Question is working -> lets do Question 3
[ 1529.836510] Current Process Id => 2374
[ 1529.836510] Parent Process Id => 2360
gp@gp-VirtualBox:~$
```

Thus successfully the messages are printed.

For Q3 are both process ids same or different ?
Ans : As we can see both the process ids are different.

Why ?
Ans : This is because one returns the Child process id and the other returns the Parent process id. Child process is the current process that is printing these values to the ring buffer. The parent will be either the process that created our current process or if that process has already terminated the process to which this terminated process has been re-parented.

What are your observations?
Ans : getpid and the abhiramgetpid system calls returns the pid of the parent process at the time of the call. If a process is reparented the value is returned accordingly. But when we print current->pid it returns the process id that is printing the value whereas the current->parent->pid returns the process id of the parent process, the terminal which invoked this systemcall.