Kushagra Gupta

2019056

Task 2:

1. Updating the kernel to 5.9.1

➔ Download the .tar file and extract it anywhere in your linux environment.

➔ Now before compiling the kernel install the necessary packages

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

➔ Now we will change the directory to new kernel code.

➔ Now we will configure our kernel

```
make menuconfig
```

A window will be popped up after this command so exit from there by using left-right keys

➔ Now we can compile our kernel by using the following command.

```
sudo make –j2
```

Here jn depends upon your linux machine where parameter n is the number of cores. In my case that n was 2 as I was using Virtual Machine so I had given 2 cores to vm.

If you have more cores you should use them as it will decrease your compilation time. This compilation took 5 hours in my case.

➔ After successful compilation type the following command

```
make modules_install install
```

This will take some time, around half an hour.

➔ After successful compilation of the last command, we need to reboot our linux

```
sudo reboot
```

➔ Now check your kernel version by typing the following command

```
uname –r
```

2. <u>Adding and testing a System call</u>

➔ I have added the system call directly in to the sys.c file which is present in kernel directory before the endif keyword.

Path of the sys.c

linux5.9.1/kernel/

➔ Now go to the linux5.9.1/arch/x86/entry/syscalls_64.tbl and add new pid.

I have added it at 335.

➔ Now we need to compile it again, type the following commands

```
sudo make –j2
make modules_install install
sudo reboot
```

We have added our system call successfully.

➔ Now to test the system call we will write a code in c, in my case I have written it in test.c file.

➔ Now compile and execute it by the following commands

```
gcc test.c –o test
./test 1
```

```
aman@aman:~/Desktop/Question2$ make
gcc test.c
aman@aman:~/Desktop/Question2$ make run
./a.out
Enter PID : 1
'sh_task_info' executed successfully
aman@aman:~/Desktop/Question2$
```

After successful execution we can check kernel log via ‘sudo dmesg’ command.

```
[ 3406.088185] PID Number: 1
[ 3406.088187] Process: systemd
[ 3406.088188] Process State: 1
[ 3406.088189] Priority: 120
[ 3406.088190] Exit State: 0
[ 3406.088191] Exit Code: 0
[ 3406.088191] Pdeath Signal: 0
aman@aman:~/Desktop$
```

3. Error Handling:

➔ If the user enters pid less than or equal to 0 or greater than 32768, the function gives an error invalid argument and also error no. 22.

```
aman@aman:~/Desktop/Question2$ make run
./a.out
Enter PID : 99999
Error : Invalid argument
'sh_task_info' didn't execute with Error No.: 22
aman@aman:~/Desktop/Question2$
```

➔ If user enters an invalid command other than integers then our function gives an invalid argument error and also error no. 22.

```
aman@aman:~/Desktop/Question2$ make run
./a.out
Enter PID : abc
Error : Invalid argument
'sh_task_info' didn't execute with Error No.: 22
aman@aman:~/Desktop/Question2$
```