

Introduction to Operating Systems

Project 2: Linux kernel system calls

Deadline:

2020-11-27 (Fri) 23:59:59

Q&A:

If you have any questions, please post it on the E3 discussion board, and it will be answered within two days.

Deliverables:

1. **Demo video** (3-5 minutes – upload to YouTube – add link in the first page of the report).
2. **Report** (pdf file with file name of the form **OS_Project02_StudentID.pdf**)
 - Screenshots + one explanation paragraph per screenshot.
 - Answers to questions below.
3. **Files: (To be uploaded to E3)**
 - a. `numericalTest.c`
 - b. `syscallsNumerical.c`
 - c. `syscall_64.tbl`
 - d. `syscalls.h`
4. In the demo video and report, for each screenshot, explain:
 - a. What has been done, and
 - b. The reasoning behind the steps.
5. In the c code, please keep correct indentation, clean code and clear comments.

Objective: The objective of this project is to help the student to get familiar with system calls. The student will learn the definition, usage and how to implement custom system calls.

Scope: Understand the concept of system call, and implement custom calls.

Questions to be answered in the report:

1. What is Kernel space? What is user space? What are the differences between them?
2. What are protection rings? How many are them? What is Ring 0? What is Ring 1?
3. What is a system call? How many types are they in total? What are the differences between all the types?
4. For the custom kernel built in project 1A, where is the list of system calls? (Give the file name and path) (Hint: Check project 1B)
5. What is the system call ID?
6. What do the reserved words “*asm linkage*” and “*printk*” mean? What does the command *update-initramfs* do?
7. How do you use **printk**? How do you read the messages printed by **printk**?
8. What is the **kernel ring buffer**? How do you read its contents?
9. What is a function signature?
10. What does **SYSCALL_DEFINE[n]** mean? What is **n**?

11. For a system call wrapper (SYSCALL_DEFINE), how does its function signature look like when it has 0 inputs as parameters? 1 integer number as input? 2 integer numbers as inputs? 3 integer numbers as inputs?
12. Does the function signature of a SYSCALL_DEFINE wrapper change depending on the type of element returned?
13. What is **#include <linux/kernel.h>**? What is **#include <linux/syscalls.h>**?

SECTION 1: SYSTEM CALLS

Section 1: System calls

Section 1.1: System requirements

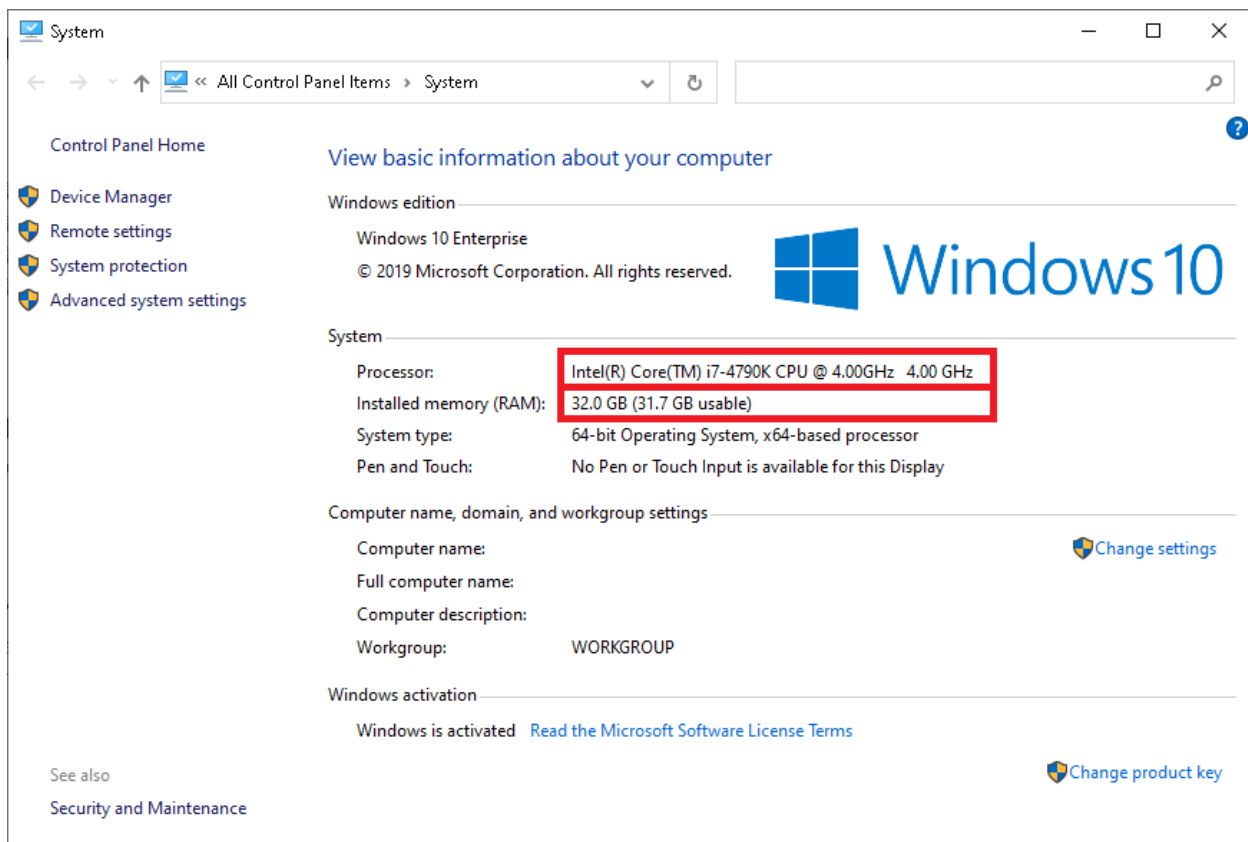
Below are the system requirements your system must match to continue:

1. Ubuntu 16.04 - 64 bits – desktop edition.
2. Kernel 4.19.148.

<https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.19.148.tar.xz>

Also check your RAM-CPU in both your physical computer and virtual machines:

1. Check your CPU model (intel i3-xxxx, intel i5-xxxx, intel i7-xxxx or similar), and check how many cores and how many threads it supports.



(If it is an intel CPU, check on ark.intel.com).

2. If your computer has only 2 cores with at most 2 threads:
 - a. **Case 1:** 2 cores – 1 thread or
 - b. **Case 2:** 2 cores – 2 threads,

Then only allocate 1 core per virtual machine and leave 2 cores for the physical machine.

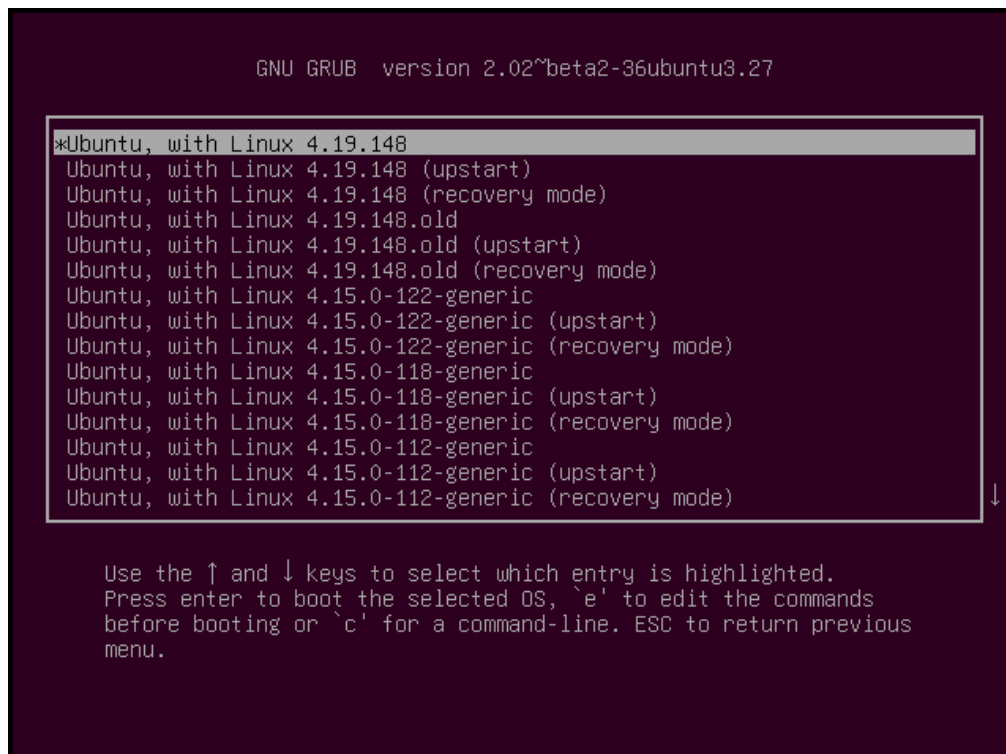
If your physical computer has more cores (i.e., $4 < \text{CPUs} = \text{\#of cores} * \text{\#of threads}$), then you can allocate more CPUs to the virtual machines.

3. If your computer only has between 4 and 6 GB of RAM, then only allocate 1-2 GB of RAM for each virtual machine and leave 2-4 GB of RAM to the physical machine.

If your computer has more than 6 GB of RAM, then you can proceed to allocate more RAM to the virtual machines.

IMPORTANT NOTES:

1. If your computer does not cover the minimum requirements, **please solve them before posting any questions in the forum.**
2. Forum support will be restricted to only **kernel 4.19.148**. (Other kernels and other systems are not supported in the forum – you can still use other kernels but no forum support will be given).
3. Forum support will be restricted to **system-calls related questions only**. (If you have **vmware-related**, **linux-related** or **c-programming** related questions, please search online – **no support will be given in the forum for these topics**).
4. If you have problems booting up the virtual machines, and/or downloading or building the kernel, please refer to project 1A.
5. Please always select the correct kernel when booting up your virtual machine.



And always check by using **\$ uname -r**.

Section 1.2: Basic system call – example

In this section, we will create a basic system call that displays a custom message from Kernel. All these steps will be performed in the Target machine. The host machine will be off during this project.

NOTE: You need to finish project 1A and 1B in order to do this one.

1. In the kernel folder (4.19.148), create a folder named “**systemCallTests**”. Inside this new folder, create a folder named “**echoTest**”.
2. Create an **echoTest.c** file inside the **echoTest** folder, with the following content:

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

SYSCALL_DEFINE0(syscalltest_helloworld)
{
    printk("Hello world from a system call! OS_Project02!\n");
    return 0;
}

SYSCALL_DEFINE1(syscalltest_echo, int, studentId)
{
    printk("My student id is : [%d]\n", studentId);
    return 0;
}
```

3. Inside the **echoTest** folder, create a Makefile, and add this line to it:
obj-y := echoTest.o

[Screenshot # 1: Create a screenshot showing the folder, and the contents of both echoTest.c and Makefile]

4. Go to the Makefile in the linux-4.19.148 folder and open it.
Search for this line (it may have different folders than the line below, but it should look similar to this):

```
core-y += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/
```

and append **systemCallTests/echoTest/** to the end of it, so it looks like

```
core-y += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/
systemCallTests/echoTest/
```

5. Still inside the linux-4.19.148 folder, go to **arch/x86/entry/syscalls/** and open the **syscall_64.tbl** file.
(This step assumes that the last line in this file had number 560).
Add the following line at the end of the file.

```
561    common        syscalltest_helloworld    __x64_sys_syscalltest_helloworld
562    common        syscalltest_echo          __x64_sys_syscalltest_echo
```

[Screenshot #2 and #3: Create a screenshot showing the before and after of the syscall 64.tbl file]

6. Go to the **linux-4.19.148/include/linux** folder. Add the signature of your system call at the end of the **syscalls.h** file:

```
asmlinkage long syscalltest_helloworld(void);
asmlinkage long syscalltest_echo(int);
```

immediately before the **#endif** line, so it will look like

```
asmlinkage long syscalltest_helloworld(void);
asmlinkage long syscalltest_echo(int);

#endif
```

[Screenshot #4: Create a screenshot showing the added lines in the syscalls.h file]

7. **(This step asks you to run Project 1A Section 3 again – check project 1A for reference)**

Run again the following commands:

```
make menuconfig
make -j $(nproc)
make modules_install
make install
```

After all these are done, check again the following files (check project 1A if you do not know where to find them):

- a. **initrd.img-4.19.148**
- b. **vmlinuz-4.19.148**
- c. **system.map.4.19.148**
- d. **config-4.19.148**

These files should be updated.

Check **initrd.img-4.19.148**. It should be around 500 Mb (if it is bigger it is still ok). If the size is significantly smaller, that means that there was an error in one of the make steps. Re-do them and check where the error is **and fix it (if you have questions, check online)**.

8. Restart the target machine.
9. In your desktop, create a folder called **SystemCallsRunTests**, and inside a subfolder called **echo**.

10. Inside the echo folder, create the following program and call it **syscallsHelloEco.c**

```
#include <stdio.h>
#include <unistd.h>
#include <sys/syscall.h>

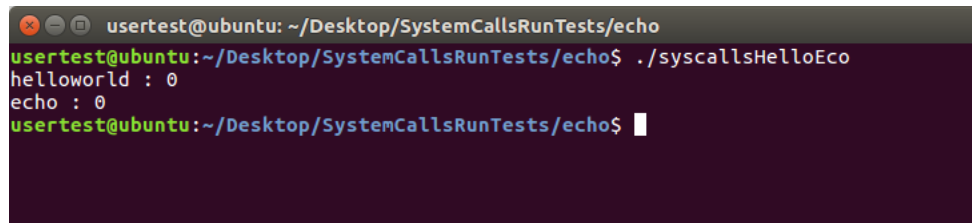
int main()
{
    int studentId = 12345;

    printf("helloworld : %ld\n", syscall(561));
    printf("echo : %ld\n", syscall(562, studentId));

    return 0;
}
```

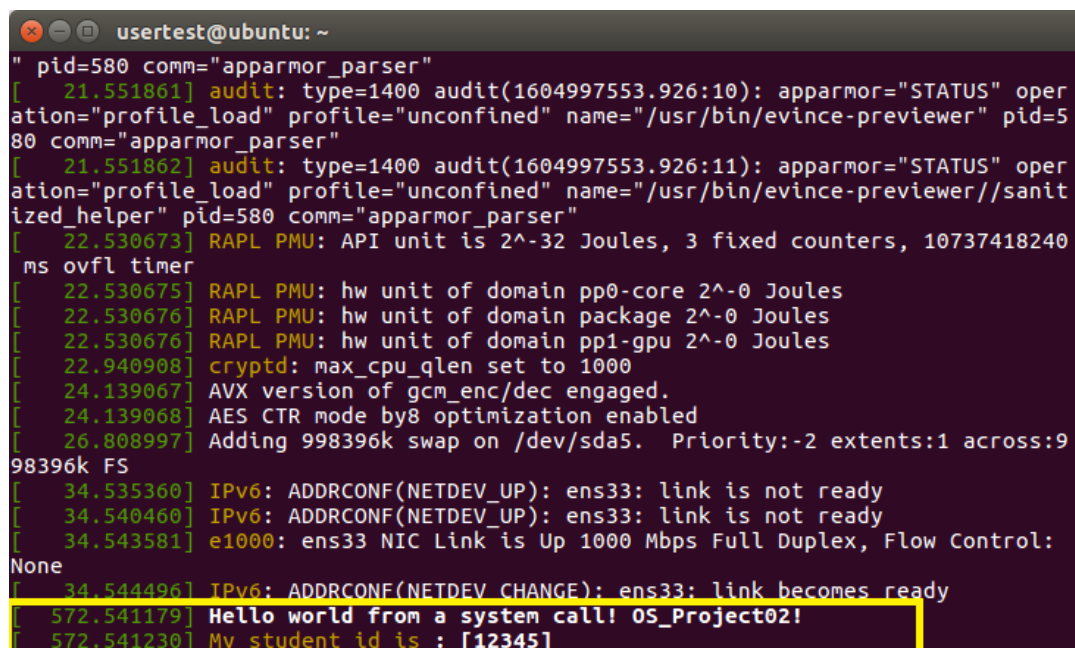
(Check step 5 for the correct system call ID. Also **replace 12345 with your student ID**)

11. Build and run the program (no special gcc attribute is required) – refer to project 1B or check online to see how to build c programs in Linux.
12. You should see an output like the one below.



```
userstest@ubuntu: ~/Desktop/SystemCallsRunTests/echo
userstest@ubuntu:~/Desktop/SystemCallsRunTests/echo$ ./syscallsHelloEco
helloworld : 0
echo : 0
userstest@ubuntu:~/Desktop/SystemCallsRunTests/echo$
```

13. In order to see the message with your student ID, run the command
\$ dmesg



```
userstest@ubuntu: ~
" pid=580 comm="apparmor_parser"
[ 21.551861] audit: type=1400 audit(1604997553.926:10): apparmor="STATUS" operation="profile_load" profile="unconfined" name="/usr/bin/evince-previewer" pid=580 comm="apparmor_parser"
[ 21.551862] audit: type=1400 audit(1604997553.926:11): apparmor="STATUS" operation="profile_load" profile="unconfined" name="/usr/bin/evince-previewer//sanitized_helper" pid=580 comm="apparmor_parser"
[ 22.530673] RAPL PMU: API unit is 2^-32 Joules, 3 fixed counters, 10737418240 ms ovfl timer
[ 22.530675] RAPL PMU: hw unit of domain pp0-core 2^-0 Joules
[ 22.530676] RAPL PMU: hw unit of domain package 2^-0 Joules
[ 22.530676] RAPL PMU: hw unit of domain pp1-gpu 2^-0 Joules
[ 22.940908] cryptd: max_cpu_qlen set to 1000
[ 24.139067] AVX version of gcm_enc/dec engaged.
[ 24.139068] AES CTR mode by8 optimization enabled
[ 26.808997] Adding 998396k swap on /dev/sda5. Priority:-2 extents:1 across:998396k FS
[ 34.535360] IPv6: ADDRCONF(NETDEV_UP): ens33: link is not ready
[ 34.540460] IPv6: ADDRCONF(NETDEV_UP): ens33: link is not ready
[ 34.543581] e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: None
[ 34.544496] IPv6: ADDRCONF(NETDEV_CHANGE): ens33: link becomes ready
[ 572.541179] Hello world from a system call! OS_Project02!
[ 572.541230] My student id is : [12345]
```

[Screenshot #5: Create a screenshot showing the dmesg output with your student ID]

Section 1.3: Basic system call – custom implementation

1. Download the **numericalTest.c** file from E3.

```
//STUDENT ID: 12345

#include <linux/syscalls.h>
#include <linux/kernel.h>

/* Insert your code in this file. You must create 4 system calls:
 * 1. syscalltest_returnIndividualValues:
 *    - Input: integers studentId, a and b.
 *    - Output: 0;
 *    - Action: Display in kernel ring buffer the message "[studentId]
syscalltest_returnIndividualValues : a, b
 *    - Example: if studentId = 12345, a=2 and b=30, then the message "[12345]
syscalltest_returnIndividualValues : 2, 30" is display in kernel ring buffer.

 * 2. syscalltest_addition:
 *    - Input: integers studentId, a and b.
 *    - Output: a+b;
 *    - Action: Display in kernel ring buffer the message "[studentId]
syscalltest_addition : a, b, a+b
 *    - Example: if studentId = 12345, a=10 and b=35, then the message "[12345]
syscalltest_addition : 10, 35, 45" is display in kernel ring buffer.

 * 3. syscalltest_multiplication:
 *    - Input: integers studentId, a and b.
 *    - Output: a*b;
 *    - Action: Display in kernel ring buffer the message "[studentId]
syscalltest_multiplication : a, b, a*b
 *    - Example: if studentId = 12345, a=11 and b=35, then the message "[12345]
syscalltest_multiplication : 11, 35, 385" is display in kernel ring buffer.
 *
 * 4. syscalltest_dataTypes:
 *    - Input: integer studentId.
 *    - Output: 0;
 *    - Action: Display the size in bits of the following datatypes:
        a. unsigned int
        b. signed int
        c. unsigned long
        d. signed long
        e. unsigned long long
        f. signed long long
        g. double
        h. char
 *    - Example: if studentId = 12345, then the message "[12345] Size of unsigned int :
4 bytes" is display in kernel ring buffer. (Do this for all the datatypes)
 *    - NOTE: You cannot hard-type "4 bytes" or "8 bytes". You need to find out how to
measure the size of the datatypes.
 */

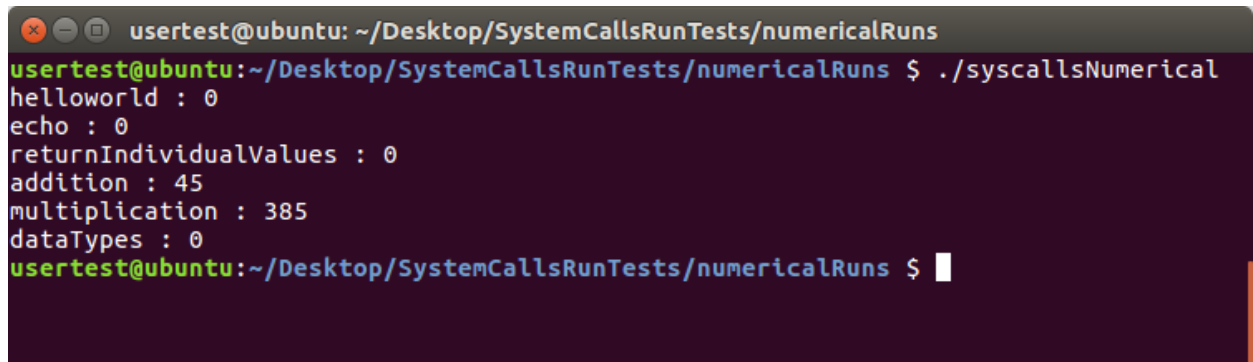
//INSERT YOUR CODE BELOW
```

2. Create a folder named **numericalTest** inside the **systemCallTests** (step 1 section 1.2) and save the **numericalTest.c** there.
3. Finish writing the **numericalTest.c** program for these four system calls:
 - a. `syscalltest_returnIndividualValues`
 - b. `syscalltest_addition`
 - c. `syscalltest_multiplication`
 - d. `syscalltest_dataTypes`

as specified in the comments inside that file.

[Screenshot #6: Create a screenshot showing the completed system calls inside the **numericalTest.c** file and EXPLAIN EACH ONE OF THEM IN DETAILS. – Write 4 small paragraphs – one per each function]

4. Complete all the needed steps to add these new system calls to your kernel.
[Screenshot #7-8-9-10: Create at least 4 screenshots showing how you do this - one of the screenshots must show the linux Makefile]
5. Inside the **SystemCallsRunTests** folder (Step 9 – Section 1.2), create a folder named **numericalRuns**. Write a program (by yourself) to test your four system calls and call it **syscallsNumerical.c**. This program should be similar to **syscallsHelloEco.c** and it must have an output similar to the following one.



```
usertest@ubuntu: ~/Desktop/SystemCallsRunTests/numericalRuns
usertest@ubuntu:~/Desktop/SystemCallsRunTests/numericalRuns $ ./syscallsNumerical
helloworld : 0
echo : 0
returnIndividualValues : 0
addition : 45
multiplication : 385
dataTypes : 0
usertest@ubuntu:~/Desktop/SystemCallsRunTests/numericalRuns $
```

And also generate a kernel ring buffer message like the one below.

```
userstest@ubuntu: ~/Desktop/Tests/Syscalls/02
file_load" profile="unconfined" name="/usr/bin/evince" pid=573 comm="apparmor_parser"
[ 19.140770] audit: type=1400 audit(1604887167.177:11): apparmor="STATUS" operation="prof
file_load" profile="unconfined" name="/usr/bin/evince//sanitized_helper" pid=573 comm="appar
mor_parser"
[ 19.800173] RAPL PMU: API unit is 2^-32 Joules, 3 fixed counters, 10737418240 ms ovfl ti
mer
[ 19.800174] RAPL PMU: hw unit of domain pp0-core 2^-0 Joules
[ 19.800175] RAPL PMU: hw unit of domain package 2^-0 Joules
[ 19.800175] RAPL PMU: hw unit of domain pp1-gpu 2^-0 Joules
[ 19.995066] cryptd: max_cpu_qlen set to 1000
[ 20.064707] AVX version of gcm_enc/dec engaged.
[ 20.064708] AES CTR mode by8 optimization enabled
[ 20.924107] Adding 998396k swap on /dev/sda5. Priority:-2 extents:1 across:998396k FS
[ 25.940655] IPv6: ADDRCONF(NETDEV_UP): ens33: link is not ready
[ 25.948345] IPv6: ADDRCONF(NETDEV_UP): ens33: link is not ready
[ 25.951194] e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: None
[ 25.952547] IPv6: ADDRCONF(NETDEV_CHANGE): ens33: link becomes ready
[ 3286.714298] Hello world from a system call! OS_Project02!
[ 3286.714377] My student id is : [12345]
[ 3286.714382] [12345] syscalltest_returnValues : 2, 30
[ 3286.714385] [12345] syscalltest_addition : 10, 35, 45
[ 3286.714388] [12345] syscalltest_multiplication : 11, 35, 385
[ 3286.714391] [12345] Size of unsigned int : 4 bytes.
[ 3286.714391] [12345] Size of signed int : 4 bytes.
[ 3286.714391] [12345] Size of unsigned long : 8 bytes.
[ 3286.714392] [12345] Size of signed long : 8 bytes.
[ 3286.714392] [12345] Size of unsigned long long : 8 bytes.
[ 3286.714393] [12345] Size of signed long long : 8 bytes.
[ 3286.714393] [12345] Size of double : 8 bytes.
[ 3286.714393] [12345] Size of char : 1 bytes.
```

[Screenshot #11-12-13: Create at least 3 screenshots showing the finished syscallsNumerical.c file, its execution results and the kernel ring buffer after its execution.]

IMPORTANT NOTES:

1. **Besides your report and video**, you must upload the following files to E3:
 - a. numericalTest.c
 - b. syscallsNumerical.c
 - c. syscall_64.tbl
 - d. syscalls.h

Keep your code clean, properly indented and well commented.

2. In **numercialTest.c** and **syscallsNumerical.c**, the first line must be your student ID:

```
//STUDENT ID: 12345
```

3. You cannot hard-code the outputs of the system calls. Random integers will be used to test your code.
4. You cannot hard-code the outputs of the datatypes' sizes (**syscalltest_dataTypes**). Search online how to get their sizes in runtime (hard-coding their values will get 0 points).
5. If you have vmware-related, Linux-related or c-related questions, please check online.

No support will be given in the forum for these type of questions.