ECE 8400 / ENGI 9875 Lab 2: system calls

Mohammed Shakeel, Tousif Habib

TOTAL POINTS

28.5 / 43

QUESTION 1

Pre-lab 13 pts

1.1 Manual pages 2 / 3

- **0 pts** Correct

✓ **- 1 pts** *Write(1) writes to the terminal(stdout) of other users , write(2) is a system call to write in files*

- **1 pts** Manual section 2 describes system calls, and section 3 describes library functions

- **3 pts** Not submitted

1.2 strace(1) 0 / 2

- **0 pts** Correct

- **1 pts** Incomplete explanation

✓ **- 2 pts** *Not answered*

1.3 unistd_64.h 0 / 5

- **0 pts** Correct

- **1 pts** Wrong file is chosen or not clearly chosen

- **2 pts** Missing the part including checking the OS version, and clear reasoning

- **2 pts** Poor reasoning

- **1 pts** Missing evidence of the work

- **1 pts** Incorrect system call codes

✓ **- 5 pts** *Not answered*

1.4 Registers 3 / 3

✓ **- 0 pts** *Correct*

- **1 pts** Missing system call number

- **1 pts** Incorrect/Incomplete information for sysenter

- **3 pts** Not answered

QUESTION 2

System call tracing 6 pts

2.1 echo "hello" 3 / 3

✓ **- 0 pts** *Correct*

- **2 pts** Poor description

- **2.5 pts** The evidence of the work has shown, but not/poorly explained.

2.2 echo "hello" vs cat /etc/fstab 3 / 3

✓ **- 0 pts** *Correct*

- **1 pts** Incomplete explanation

- **2.5 pts** The evidence of the work has shown, but not/poorly explained.

QUESTION 3

System call assembly code 12 pts

3.1 Object file 2 / 2

✓ **- 0 pts** *Correct*

- **2 pts** Not answered

- **0.5 pts** Missing minor details

- **1 pts** Missing major differences: information about the addresses, and symbols replaced by the

address pointer of the variables

- 1.5 pts The evidence of the work is provided, but no explanation

3.2 Executable 3 / 3

✓ **- 0 pts** *Correct*

- 3 pts Not answered

- 2 pts Failed to explain the major differences in the function calls and the moved values

- 2.5 pts Provided the evidence of the work, but no explanation

3.3 Trace 2 / 2

✓ **- 0 pts** *Correct*

- 2 pts Not answered

- 1 pts Failed to address the main difference: an undetected system call

3.4 Interpretation 3 / 5

- 0 pts Correct

- 5 pts Not answered

✓ **- 2 pts** *Failed to explain how the system call matches. (It does not match any system call, because it's addressing the "message" variable!)*

- 4 pts Showed the work, but no explanation

QUESTION 4

Invoking system calls 12 pts

4.1 Print your names 3 / 3

✓ **- 0 pts** *Correct*

- 1 pts The program execution and output are not shown.

- 1 pts Incomplete explanation

- 1 pts The complete code is not provided

4.2 syscall 1.5 / 2

- 0 pts Correct

✓ **- 0.5 pts** *The program execution and the output are not shown*

- 0.5 pts The modified code is not provided

- 0.5 pts Incomplete/poor explanation

- 1 pts No explanation

4.3 Timing 3 / 3

✓ **- 0 pts** *Correct*

- 3 pts Not answered

- 1 pts The code or the program execution's output are not provided

- 1 pts No analysis and comparison provided between the two versions

- 0 pts Click here to replace this description.

4.4 Read and write 0 / 4

- 0 pts Correct

✓ **- 4 pts** *Not answered*

- 1 pts Incomplete code

- 1 pts Incomplete explanation

- 1 pts Missing the evidence of the program execution and the output

1.1 Manual pages **2 / 3**

- **0 pts** Correct

✓ **- 1 pts** *Write(1) writes to the terminal(stdout) of other users , write(2) is a system call to write in files*

- **1 pts** Manual section 2 describes system calls, and section 3 describes library functions

- **3 pts** Not submitted

1.2 strace(1) **0 / 2**

- 0 pts Correct

- 1 pts Incomplete explanation

✓ **- 2 pts** *Not answered*

1.3 unistd_64.h **0 / 5**

- **0 pts** Correct
- **1 pts** Wrong file is chosen or not clearly chosen
- **2 pts** Missing the part including checking the OS version, and clear reasoning
- **2 pts** Poor reasoning
- **1 pts** Missing evidence of the work
- **1 pts** Incorrect system call codes

✓ **- 5 pts** *Not answered*

1.4 Registers **3 / 3**

✓ **- 0 pts** *Correct*

- 1 pts Missing system call number

- 1 pts Incorrect/Incomplete information for sysenter

- 3 pts Not answered

2.1 echo "hello" **3 / 3**

✓ **- 0 pts** *Correct*

- 2 pts Poor description

- 2.5 pts The evidence of the work has shown, but not/poorly explained.

2.2 echo "hello" vs cat /etc/fstab **3 / 3**

✓ **- 0 pts** *Correct*

- 1 pts Incomplete explanation

- 2.5 pts The evidence of the work has shown, but not/poorly explained.

3.1 Object file **2 / 2**

✓ **- 0 pts** *Correct*

- 2 pts Not answered

- 0.5 pts Missing minor details

- 1 pts Missing major differences: information about the addresses, and symbols replaced by the address pointer of the variables

- 1.5 pts The evidence of the work is provided, but no explanation

3.2 Executable **3 / 3**

✓ **- 0 pts** *Correct*

- 3 pts Not answered

- 2 pts Failed to explain the major differences in the function calls and the moved values

- 2.5 pts Provided the evidence of the work, but no explanation

3.3 Trace **2 / 2**

✓ **- 0 pts** *Correct*

- 2 pts Not answered

- 1 pts Failed to address the main difference: an undetected system call

3.4 Interpretation **3 / 5**

- **0 pts** Correct

- **5 pts** Not answered

✓ **- 2 pts** *Failed to explain how the system call matches. (It does not match any system call, because it's addressing the "message" variable!)*

- **4 pts** Showed the work, but no explanation

4.1 Print your names **3 / 3**

✓ **- 0 pts** *Correct*

- 1 pts The program execution and output are not shown.

- 1 pts Incomplete explanation

- 1 pts The complete code is not provided

4.2 syscall **1.5 / 2**

- **0 pts** Correct

✓ **- 0.5 pts** *The program execution and the output are not shown*

- **0.5 pts** The modified code is not provided
- **0.5 pts** Incomplete/poor explanation
- **1 pts** No explanation

4.3 Timing **3 / 3**

✓ **- 0 pts** *Correct*

- 3 pts Not answered

- 1 pts The code or the program execution's output are not provided

- 1 pts No analysis and comparison provided between the two versions

- 0 pts Click here to replace this description.

4.4 Read and write **0 / 4**

- **0 pts** Correct

✓ **- 4 pts** *Not answered*

- **1 pts** Incomplete code
- **1 pts** Incomplete explanation
- **1 pts** Missing the evidence of the program execution and the output

Lab 2

Prelab:

1. To see the manual page for the man command, which shows how to use various commands and utilities on a Linux or Unix-based system, you can use the command man 1 man. The manual page gives you detailed information about a command's options, arguments, and usage. The man command helps you learn more about the commands and utilities that are available on the system and how to use them.

2. Section 2 system calls Section 3 - a. The Linux system calls are described in section 2 of the manual. A system call is a way to access the Linux kernel's functionality. Usually, you don't call system calls directly: instead, you use C library functions that do the necessary steps (e.g., switching to kernel mode) to call the system call for you. So calling a system call looks like calling a normal library function. b. All library functions except for the ones (system call wrappers) in section 2 that call system calls are described in section 3 of the manual.

3. In C programming, write(1) and write(2) both mean calling the write system call, but with different file descriptor numbers.

   a. write(1) writes to the standard output file descriptor, which is usually where you see your program's output on the console or terminal window.

   b. write(2) writes to the standard error file descriptor, which is also usually where you see your program's output on the console or terminal window, but it is usually used for error messages or debugging output.

   When you call write() without giving a file descriptor number, it uses 1 (standard output) by default. The main difference between them is that standard output is usually used for normal program output and standard error is used for error messages and debugging output.

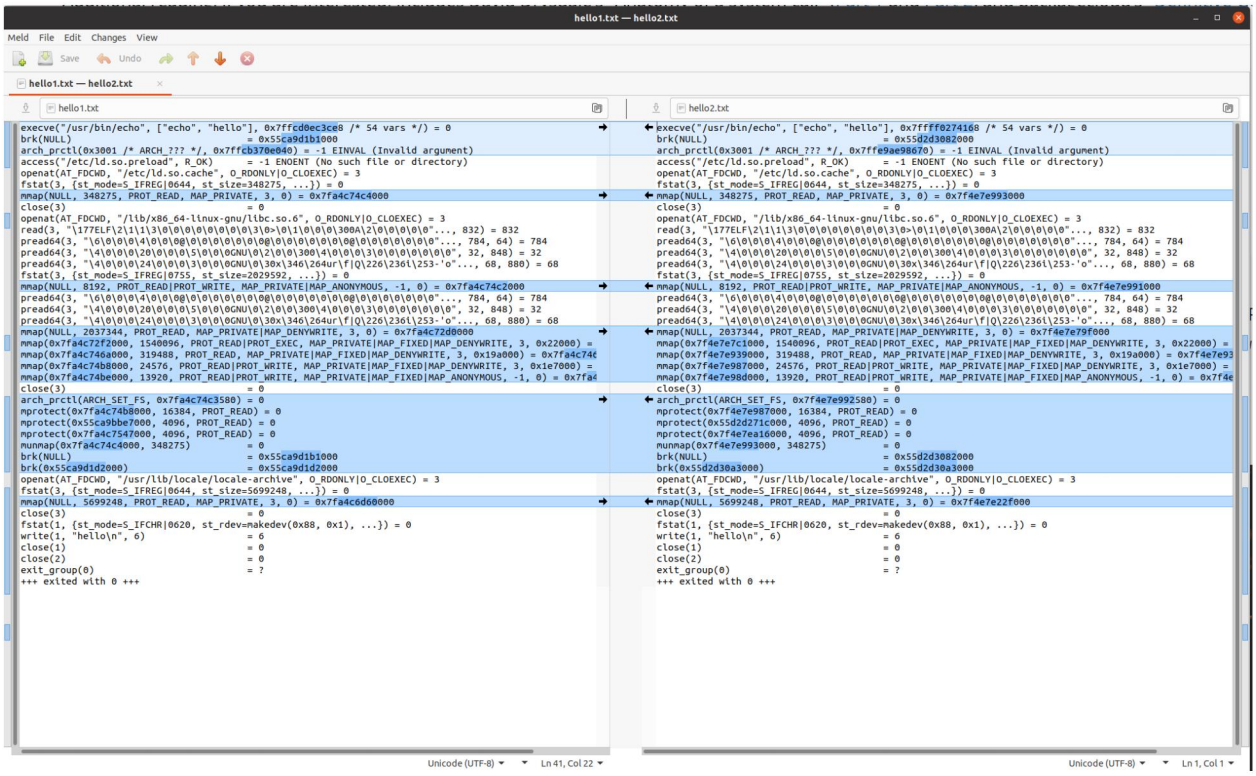
4.

	syscall	sysenter	legacy
System call number	rax	eax	eax
Argument 1	rsi	ebx	ebx
Argument 2	rdx	ecx	ecx
Argument 3	r10	edx	edx
Argument 4	r8	esi	esi
Argument 5	r9	edi	edi
Argument 6		0(%ebp)	ebp

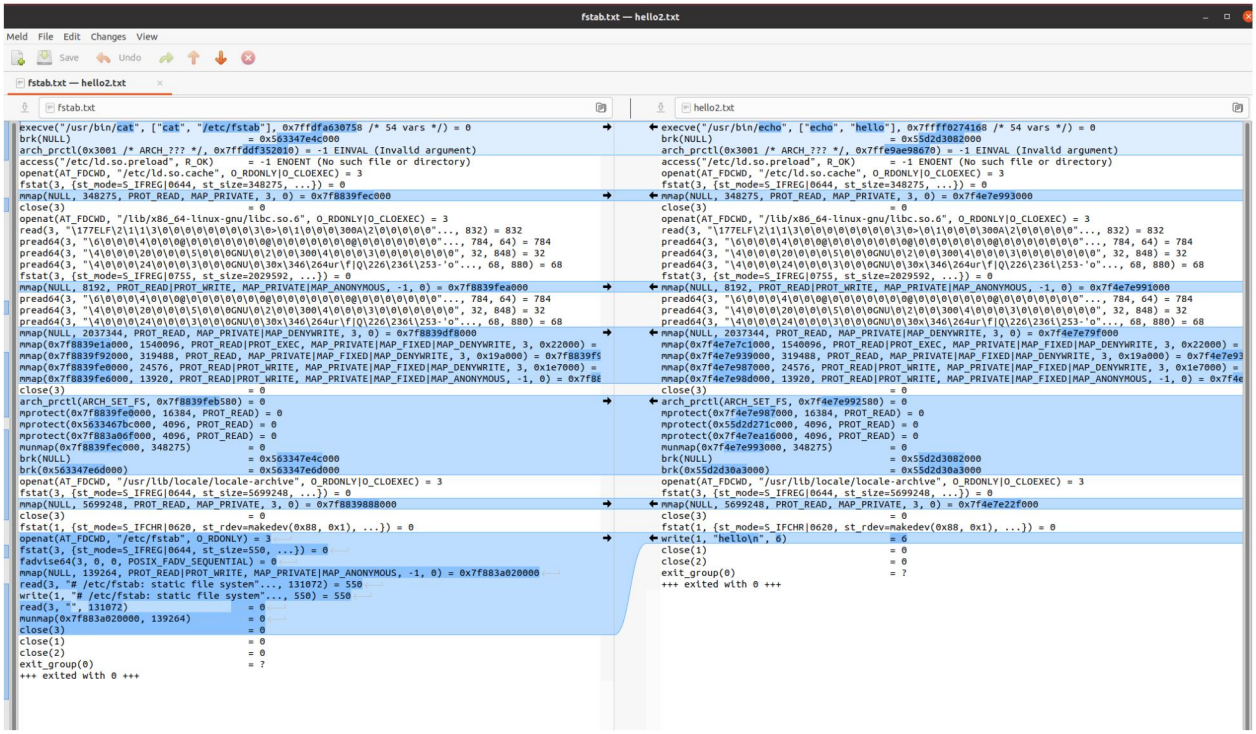
Procedure

System call tracing

1. Between the two files, the system call number is higher and the address is different.



2. The newstatat line in stab-cat.txt causes more system calls than in hello1.txt. Also, the virtual memory address is different between the two files.

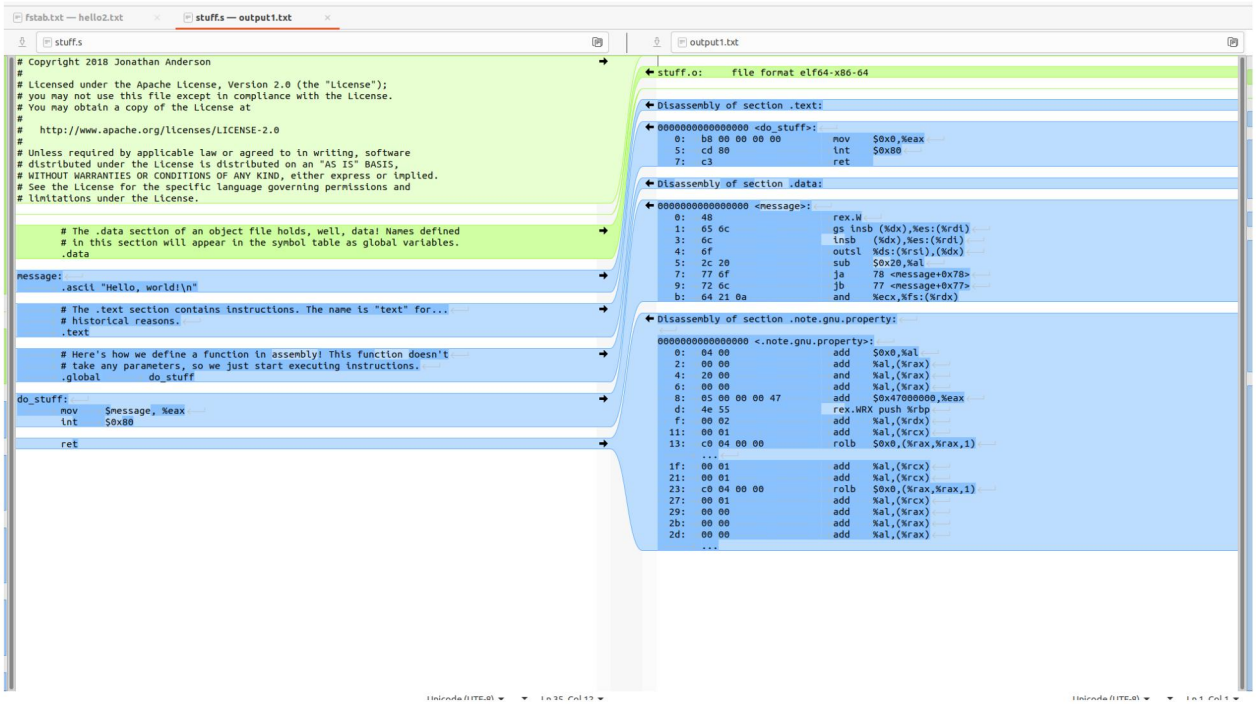


System call assembly code

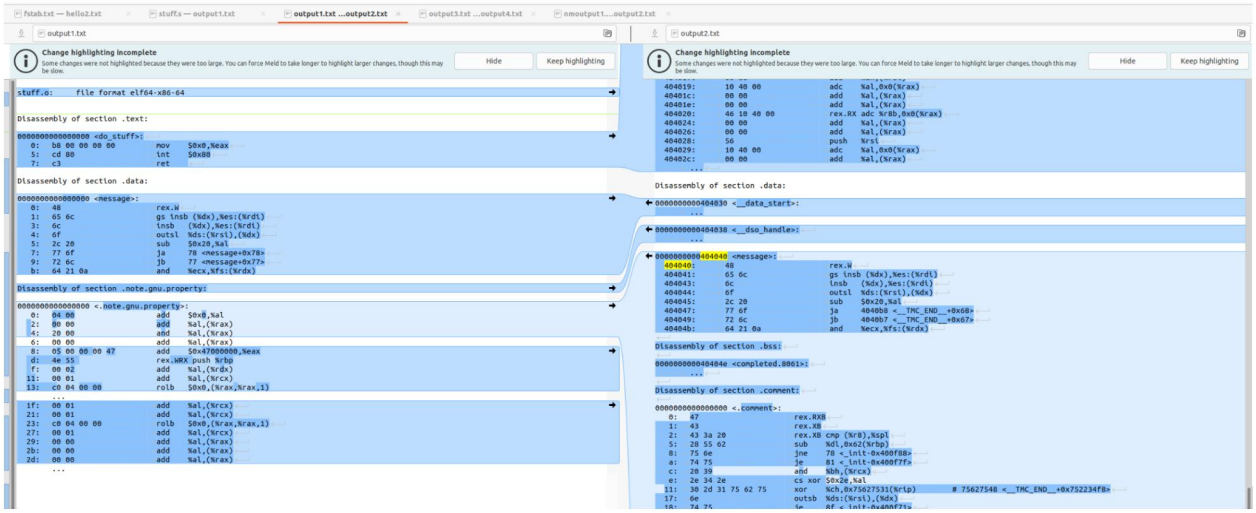
1. A comment with the message is in the .S file, while the disassembly shows addresses, opt code, and arguments along with the message.

```
msshakeel@slbncsf2112pc63:~/Labs/OSLabs/Lab2$ objdump -d stuff.o

stuff.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <do_stuff>:
   0:   b8 00 00 00 00          mov    $0x0,%eax
   5:   cd 80                   int    $0x80
   7:   c3                      ret
msshakeel@slbncsf2112pc63:~/Labs/OSLabs/Lab2$
```

```
msshakeel@slbncsf2112pc63:~/Labs/OSLabs/Lab2$ objdump -D stuff.o

stuff.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <do_stuff>:
   0:   b8 00 00 00 00          mov    $0x0,%eax
   5:   cd 80                   int    $0x80
   7:   c3                      ret

Disassembly of section .data:

0000000000000000 <message>:
   0:   48                      rex.W
   1:   65 6c                   gs insb (%dx),%es:(%rdi)
   3:   6c                      insb   (%dx),%es:(%rdi)
   4:   6f                      outsl  %ds:(%rsi),(%dx)
   5:   2c 20                   sub    $0x20,%al
   7:   77 6f                   ja     78 <message+0x78>
   9:   72 6c                   jb     77 <message+0x77>
   b:   64 21 0a                and    %ecx,%fs:(%rdx)

Disassembly of section .note.gnu.property:

0000000000000000 <.note.gnu.property>:
   0:   04 00                   add    $0x0,%al
   2:   00 00                   add    %al,(%rax)
   4:   20 00                   and    %al,(%rax)
   6:   00 00                   add    %al,(%rax)
   8:   05 00 00 00 47          add    $0x47000000,%eax
   d:   4e 55                   rex.WRX push %rbp
   f:   00 02                   add    %al,(%rdx)
  11:   00 01                   add    %al,(%rcx)
  13:   c0 04 00 00             rolb   $0x0,(%rax,%rax,1)
        ...
  1f:   00 01                   add    %al,(%rcx)
  21:   00 01                   add    %al,(%rcx)
  23:   c0 04 00 00             rolb   $0x0,(%rax,%rax,1)
  27:   00 01                   add    %al,(%rcx)
  29:   00 00                   add    %al,(%rax)
  2b:   00 00                   add    %al,(%rax)
  2d:   00 00                   add    %al,(%rax)
        ...
msshakeel@slbncsf2112pc63:~/Labs/OSLabs/Lab2$
```



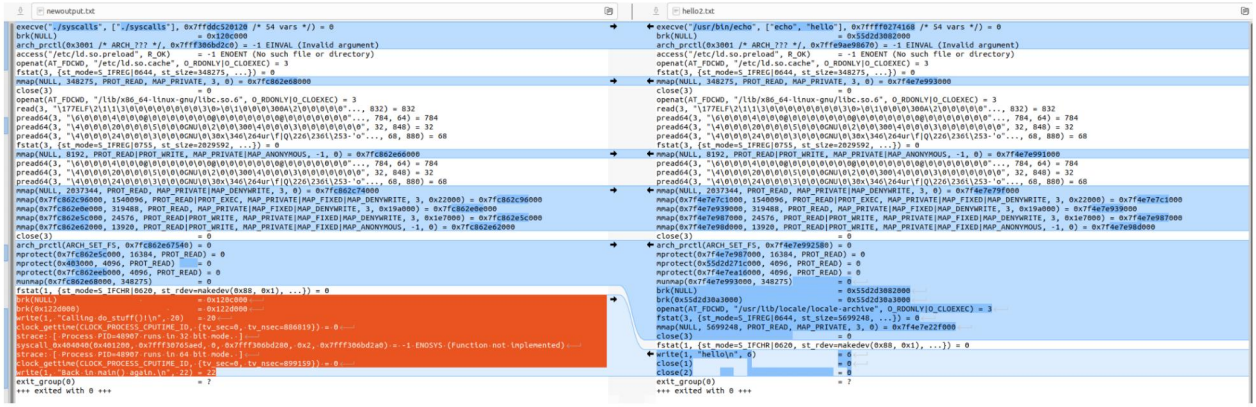
2. I also compared two functions: do_stuff and do_stuff of syscalls. I also observed that the do_stuff functions were almost identical in both files, except for an extra “xchg” op code in the syscalls version. To investigate further, I used nm2 to examine their symbols. This showed me many more differences between stuff.o and syscalls, such as .bss, main, etc., which are related to different sections of an executable file5. The only similarities were the do_stuff function and message.



```
0000000000404040 <message>:
  404040:	48                   	rex.W
  404041:	65 6c                	gs insb (%dx),%es:(%rdi)
  404043:	6c                   	insb   (%dx),%es:(%rdi)
  404044:	6f                   	outsl  %ds:(%rsi),(%dx)
  404045:	2c 20                	sub    $0x20,%al
  404047:	77 6f                	ja     4040b8 <__TMC_END__+0x68>
  404049:	72 6c                	jb     4040b7 <__TMC_END__+0x67>
  40404b:	64 21 0a             	and    %ecx,%fs:(%rdx)

Disassembly of section .bss:
```

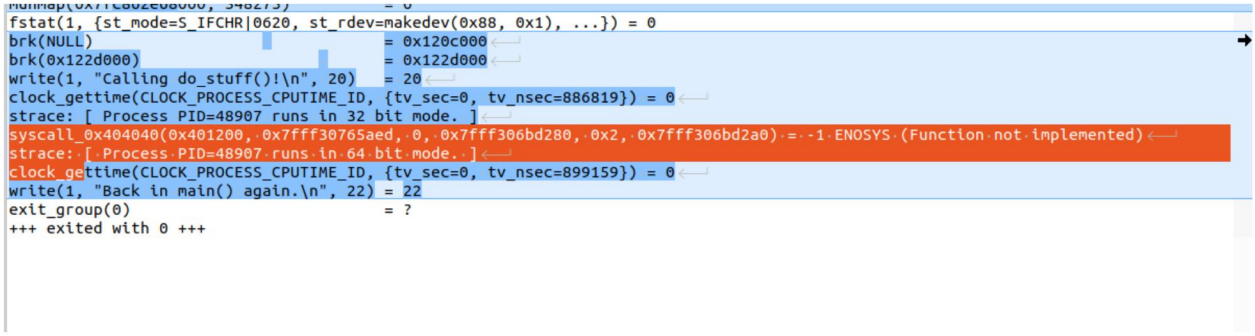
```
thabib@thabib-X541UAK:~/School/Term8/OS/labs/1$ nm stuff.o
0000000000000000 T do stuff
0000000000000000 d message
```

3. The two executions are different because they run different commands: one runs “./syscalls” and the other runs “echo”, “hello”. They also use different memory addresses as expected. In the syscall execution, there is an attempt to call a system function at 0x404040, but it fails because the function is not implemented.



4.

```
msshakeel@slbncsf2112pc63:~/Labs/OSLabs/Lab2$ lldb syscalls
(lldb) target create "syscalls"
Current executable set to '/users/labnet/st3/msshakeel/Labs/OSLabs/Lab2/syscalls' (x86_64).
(lldb) breakpoint set -b do_stuff
Breakpoint 1: where = syscalls`do_stuff, address = 0x00000000004011f7
(lldb) n
error: invalid process
(lldb) run
Process 58670 launched: '/users/labnet/st3/msshakeel/Labs/OSLabs/Lab2/syscalls' (x86_64)
Calling do_stuff()!
Process 58670 stopped
* thread #1, name = 'syscalls', stop reason = breakpoint 1.1
    frame #0: 0x00000000004011f7 syscalls`do_stuff
syscalls`do_stuff:
->  0x4011f7 <+0>: movl   $0x404040, %eax           ; imm = 0x404040
    0x4011fc <+5>: int    $0x80
    0x4011fe <+7>: retq
    0x4011ff <+8>: nop
(lldb) n
Process 58670 stopped
* thread #1, name = 'syscalls', stop reason = instruction step over
    frame #0: 0x00000000004011fc syscalls`do_stuff + 5
syscalls`do_stuff:
->  0x4011fc <+5>: int    $0x80
    0x4011fe <+7>: retq
    0x4011ff <+8>: nop

syscalls`__libc_csu_init:
    0x401200 <+0>: endbr64
(lldb) register read
General Purpose Registers:
       rax = 0x0000000000404040  message
       rbx = 0x0000000000401200  syscalls`__libc_csu_init
       rcx = 0x00007ffff7fcdaed  [vdso]`__vdso_clock_gettime + 221
       rdx = 0x0000000000000000
       rdi = 0x0000000000000002
       rsi = 0x00007fffffffe050
       rbp = 0x00007fffffffe070
       rsp = 0x00007fffffffe028
        r8 = 0x00007ffff7fc9080
        r9 = 0x0000000000000014
       r10 = 0x00000000004003ce
       r11 = 0x0000000000000246
       r12 = 0x0000000000401060  syscalls`_start
       r13 = 0x00007fffffffe160
       r14 = 0x0000000000000000
       r15 = 0x0000000000000000
       rip = 0x00000000004011fc  syscalls`do_stuff + 5
    rflags = 0x0000000000000246
        cs = 0x0000000000000033
        fs = 0x0000000000000000
        gs = 0x0000000000000000
        ss = 0x000000000000002b
        ds = 0x0000000000000000
        es = 0x0000000000000000
```



Inside rax trying to write into 404040 which is an error instead should use write(1) (value is 1)

Invoking system calls

1.

```
message:
    .ascii "Shakeel\n"


    # The .text section contains instructions. The name is "text" for...
    # historical reasons.
    .text


    # Here's how we define a function in assembly! This function doesn't
    # take any parameters, so we just start executing instructions.
    .global do_stuff


do_stuff:
    mov $7,%edx
    mov $message,%ecx
    mov $1,%ebx
    mov $4,%eax
    int $0x80
    #mov    $message, %eax
    #int    $0x80


    ret
```

We used syscall 4 because the legacy system of linux is different that means syscall 1 is something else

```
msshakeel@slbncsf2112pc63:~/Labs/OSLabs/Lab2$ find /usr/src -name unistd_32.h
find: '/usr/src/libdvd-pkg': Permission denied
/usr/src/linux-headers-5.11.0-43-generic/arch/x86/include/generated/uapi/asm/unistd_32.h
/usr/src/linux-headers-5.4.0-137-generic/arch/x86/include/generated/uapi/asm/unistd_32.h
/usr/src/linux-headers-5.15.0-58-generic/arch/x86/include/generated/uapi/asm/unistd_32.h
/usr/src/linux-headers-5.13.0-35-generic/arch/x86/include/generated/uapi/asm/unistd_32.h
/usr/src/linux-headers-5.13.0-51-generic/arch/x86/include/generated/uapi/asm/unistd_32.h
^C
msshakeel@slbncsf2112pc63:~/Labs/OSLabs/Lab2$ nano /usr/src/linux-headers-5.11.0-43-generic/arch/x86/include/generated/u
api/asm/unistd_32.h
```

```
#define __NR_restart_syscall 0
#define __NR_exit 1
#define __NR_fork 2
#define __NR_read 3
#define __NR_write 4
#define __NR_open 5
```

This means 1 is exit and 4 is write and hence the modifications in the assembly code

```
Output:
msshakeel@slbncsf2112pc63:~/Labs/OSLabs/Lab2$ ./syscalls
Calling do_stuff()!
Shakeel
Back in main() again.
```

2.

```
message:
   .ascii "Shakeel\n"

   # The .text section contains instructions. The name is "text" for...
   # historical reasons.
   .text

   # Here's how we define a function in assembly! This function doesn't
   # take any parameters, so we just start executing instructions.
   .global do_stuff

do_stuff:
   mov $0x01, %rax
   mov $0x01, %rdi
   mov $message, %rsi
   mov $0x10, %rdx
```

```
    syscall
    #mov    $message, %eax
    #int    $0x80

    ret
```

- `message:`: Define a label called “message” that points to the following data
- `.ascii "Shakeel\n"`: Store the string “Shakeel” followed by a newline character in memory
- `.text`: Start a new section that contains instructions
- `.global do_stuff`: Declare a function called “do_stuff” that can be accessed from other files
- `do_stuff:`: Define the start of the function “do_stuff”
- `mov $0x01, %rax`: Put 1 (hexadecimal) or 0x1 (decimal) in %rax register. This is the system call number for write.
- `mov $0x01, %rdi`: Put 1 (hexadecimal) or 0x1 (decimal) in %rdi register. This is the file descriptor for standard output.
- `mov $message, %rsi`: Put the memory address of “message” label in %rsi register. This is the buffer to write from.
- `mov $0x10, %rdx`: Put 16 (hexadecimal) or 0x10 (decimal) in %rdx register. This is the number of bytes to write.
- `syscall`: Trigger a system call with the parameters in %rax, %rdi, %rsi and %rdx registers. This will write “Shakeel\n” to standard output.
- `ret`: Return from the current function

3.

```c
#include <err.h>
#include <time.h>
#include <stdio.h>

extern void do_stuff(void);

int main(int argc, char *argv[])
{
    struct timespec begin, end;
    double elapsed_time;

    printf("Calling do_stuff()!\n");

    if (clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &begin) != 0)
```

```
    {
        err(-1, "Failed to get start time");
    }

    do_stuff();

    if (clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &end) != 0)
    {
        err(-1, "Failed to get end time");
    }

    elapsed_time = (end.tv_sec - begin.tv_sec) + (end.tv_nsec -
begin.tv_nsec) / 1000000000.0;
    printf("Elapsed time: %f seconds\n", elapsed_time);

    printf("Back in main() again.\n");

    return 0;
}
```

```
$ ministat -n 1000 -w 80 -C "syscall" -C "int 0x80" -x /dev/stdin
syscall
        N           Min           Max        Median           Avg
Stddev
x     1000      0.000004      0.000049      0.000006      0.000007
0.000004
+     1000      0.000004      0.000039      0.000006      0.000007
0.000003
Difference at 95.0% confidence
        1e-06 +/- 1.49771e-06
        14.2857% +/- 21387.6%
        (Student's t, pooled s = 0.00000397579)
```

The ministat output shows that the median and average execution times for both versions of do_stuff() are very close, with the syscall version being slightly slower. However, the standard deviation of the execution times for the int $0x80 version is smaller than the standard deviation for the syscall version. This suggests that the int $0x80 version has less variation in execution time.

The motivation for the introduction of the syscall instruction is likely related to improvements in the performance and security of system calls in newer processors and operating systems.