NATIONAL UNIVERSITY
OF COMPUTER & EMERGING SCIENCES
PESHAWAR CAMPUS

Problem Set:	Assignment 02	**Semester**:	Spring 2022
Points:	10		
Date Set:	See Slate	**Due Date**:	See Slate
Course:	CS220 Operating Systems	**Instructor**:	Nauman

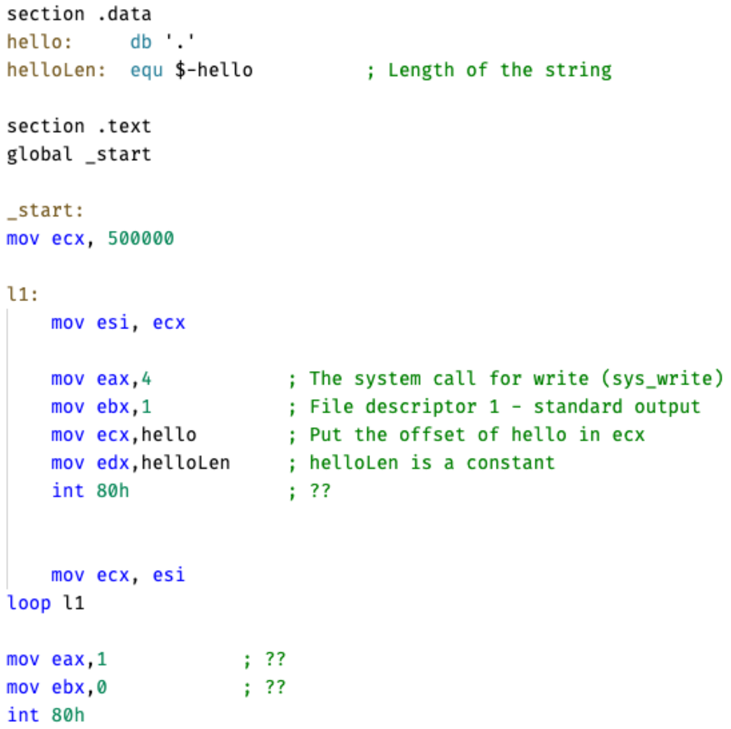
Take note of the assignment due date. Late submissions will be accepted until 48 hours after the deadline with a 2 point penalty. If you have any queries, feel free to post in Slack.

Please read the whole PDF before starting any of the tasks.

1 Tasks

The objective of this assignment is to figure out exactly how fast the new system call instructions are compared to the old method. Make sure you have a 64-bit POSIX-compliant operating system installed on your machine.

1. Write an assembly program with the following code and name it `hello.asm`. This program simply uses the `int` instruction 500,000 times to write the dot character on the console.



```
section .data
hello:     db '.'
helloLen:  equ $-hello          ; Length of the string

section .text
global _start

_start:
mov ecx, 500000

l1:
    mov esi, ecx

    mov eax,4               ; The system call for write (sys_write)
    mov ebx,1               ; File descriptor 1 - standard output
    mov ecx,hello           ; Put the offset of hello in ecx
    mov edx,helloLen        ; helloLen is a constant
    int 80h                 ; ??


    mov ecx, esi
loop l1

mov eax,1               ; ??
mov ebx,0               ; ??
int 80h
```

2. Compile the program using the command:

```
nasm -f elf64 hello.asm
```

Note: If you don't have nasm installed, you can issue the following command to install it.

```
sudo apt install nasm
```

Also issue the following if you get an error on the `ld` command below:

```
sudo apt install build-essential
```

3. Link the object file using the following command:

```
ld -s -o hello hello.o
```

4. Run the new executable:

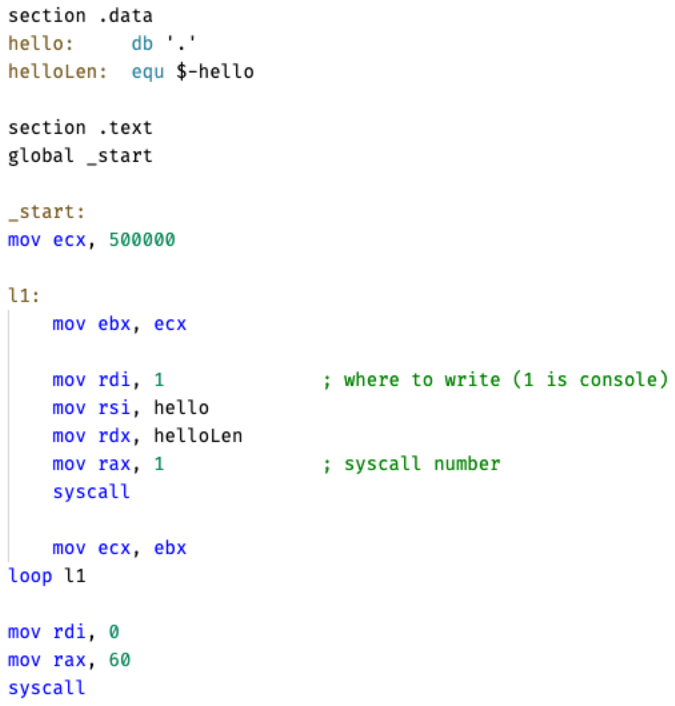
```
./hello
```

5. As you will see, this will keep going for a while since 500,000 dots are going to take a while. We're only interested in seeing how long it takes to actually issue all the interrupts. So, modify the execution command above to the following:

```
time    ./hello    > /dev/null
```

You don't have to understand how this command works but the summary is: it uses the time utility to provide precise statistics of how long the command takes and the final part ensures that any output produced by the command is essentially ignored. (You may cover this in detail in the lab or course later).

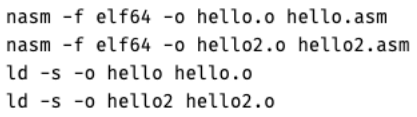
6. Now create another assembly file and name it `hello2.asm`. Put the following content in it:



```
section .data
hello:     db '.'
helloLen:  equ $-hello

section .text
global _start

_start:
mov ecx, 500000

l1:
    mov ebx, ecx

    mov rdi, 1              ; where to write (1 is console)
    mov rsi, hello
    mov rdx, helloLen
    mov rax, 1              ; syscall number
    syscall

    mov ecx, ebx
loop l1

mov rdi, 0
mov rax, 60
syscall
```

As you can see, this is the same logic but instead of the `int` instruction, we are using the `syscall` instruction which is the 64-bit equivalent of the 32-bit `sysenter` command, which in turn is the new way of doing `int` instructions. Rest of the logic should be self-evident.

Compile both of the files and link them using the following commands:



```
nasm -f elf64 -o hello.o hello.asm
nasm -f elf64 -o hello2.o hello2.asm
ld -s -o hello hello.o
ld -s -o hello2 hello2.o
```

7. You can go ahead and compile and run this code as before. However, for the sake of our original problem of finding out which one of these two is the faster one, you are provided with a runner file. This is `runner.sh`. You may look at the contents of the file.

The purpose of the runner is to run `hello` and `hello2` one by one. This pair is executed 50 times and the results of execution times of each are saved in `hello.txt` and `hello2.txt` respectively.

Make sure you have `runner.sh` saved in the same directory as your assembly files and you are in the same working directory. First, you need to make runner executable:

```
chmod +x runner.sh
```

After that, execute it:

```
./runner.sh
```

This should take a while but after it is done, you should have the two text files in the current working directory. The contents of these directories are times for each individual run. A sample looks like this:

```
real    0m0.553s
user    0m0.242s
sys     0m0.335s
```

We are interested in the `user` times. For each of the files, you need to calculate the average of the user time taken across the 50 experiments.

2 Submission

Provide screenshots of everything you have done in one PDF file. Name of the file must be your NU-ID. Also make sure you write your name, course section and ID on the first page of the PDF. If you do not follow these instructions, you may not get any credit for your work.

In your file, clearly provide the following statistics:

```
Number of experiments run:                         N =
Average ‘user time’ for hello (int-based calls):   I =
Average ‘user time’ for hello2 (syscall-based calls): S =

Percentage speedup: (I-S)*100/I                       =
```

You may use any method/tool/software you chose to calculate the averages but you need to show your working.

Submit the assignment of Slate. No other medium of submission is acceptable.

Note: Some points to consider about this assignment are: why did we issue 500k syscalls? Why not less or more? Also, why did we run the experiment 50 times? Finally, why did we have to interleave hello and hello2. Why not run hello 50 times first and then hello2 the same number of times?

Answers to these are not a part of the assignment but you should consider them nonetheless.