

Problem Set 7 :Hello World

Scott Jin

2017-12-20

Contents

1	Code Listings	1
2	Experimental Screenshots	7
3	Narrative	9
3.1	Problem 3	9

List of Figures

1	Test result for programs	7
2	Test result for straces	8
3	Test result for straces	8
4	Test result for \$?	9

1 Code Listings

Scott Jin—hello32.s

```
1 #for 32 bit
2 #worked on testing machine
3 #Linux 4acada6c5b79 4.4.0-93-generic #116~14.04.1-Ubuntu SMP Mon Aug 14 16:07:05 UTC 2017
   x86_64 x86_64 x86_64 GNU/Linux
4
5 .text
6
7 .global _start
8
9 _start:
10
11
12 # write our string to stdout.
13
14 movl    $len,%edx        # arg3: message length.
15 movl    $msg,%ecx        # arg2: pointer to message to write.
16 movl    $1,%ebx         # arg1: file handle (stdout).
17 movl    $4,%eax         # sys_write
18 int     $0x80           # syscall
19
20 # and exit.
21
22 movl    $5,%ebx         # exit code 0
23 movl    $1,%eax         # sys_exit
24 int     $0x80
25 .data
26
27 msg:
28 .ascii  "Hello, \world!\n"
29 len = . - msg          # length
```

```
1 #version for 64 bit
2 #worked on testing machine
3 #Linux 4acada6c5b79 4.4.0-93-generic #116~14.04.1-Ubuntu SMP Mon Aug 14 16:07:05 UTC 2017
   x86_64 x86_64 x86_64 GNU/Linux
4 .text
5
6 .global _start
7
8 _start:
9     # write(1, message, 13)
10    mov     $1, %rax
11    mov     $1, %rdi
12    mov     $message, %rsi
13    mov     $13, %rdx
14    syscall
15
16    # exit(10)
17    mov     $60, %rax
18    mov     $5, %rdi
19    syscall
20    message:
21    .ascii  "Hello, \world\n"
```

```

1 #for 32 bit
2 #worked on testing machine
3 #Linux 4acada6c5b79 4.4.0-93-generic #116~14.04.1-Ubuntu SMP Mon Aug 14 16:07:05 UTC 2017
   x86_64 x86_64 x86_64 GNU/Linux
4
5 .text
6
7 .global _start
8
9 _start:
10
11
12 # write our string to stdout.
13
14 movl    $len,%edx        # arg3: message length.
15 movl    $msg,%ecx        # arg2: pointer to message to write.
16 movl    $1,%ebx         # arg1: file handle (stdout).
17 movl    $4,%eax         # sys_write
18 int     $0x80           # syscall
19
20 # and exit.
21 msg:
22 .ascii  "Hello,\nworld!\n"
23 len = . - msg           # length

```

```
1 #for 32 bit
2 #worked on testing machine
3 #Linux 4acada6c5b79 4.4.0-93-generic #116~14.04.1-Ubuntu SMP Mon Aug 14 16:07:05 UTC 2017
   x86_64 x86_64 x86_64 GNU/Linux
4 .text
5
6 .global _start
7
8 _start:
9     # write(1, message, 13)
10    mov     $1, %rax
11    mov     $1, %rdi
12    mov     $message, %rsi
13    mov     $13, %rdx
14    syscall
15
16    message:
17    .ascii  "Hello, \world\n"
```

```

1  #for 32 bit
2  #worked on testing machine
3  #Linux 4acada6c5b79 4.4.0-93-generic #116~14.04.1-Ubuntu SMP Mon Aug 14 16:07:05 UTC 2017
   x86_64 x86_64 x86_64 GNU/Linux
4
5  .text
6
7  .global _start
8
9  _start:
10
11
12  # write our string to stdout.
13
14  movl    $len,%edx          # arg3: message length.
15  movl    $msg,%ecx          # arg2: pointer to message to write.
16  movl    $1,%ebx            # arg1: file handle (stdout).
17  movl    $400,%eax          # sys_write
18  int     $0x80              # syscall
19
20  # and exit.
21
22  movl    $5,%ebx            # exit code 0
23  movl    $1,%eax            # sys_exit
24  int     $0x80
25  .data
26
27  msg:
28  .ascii  "Hello, \world!\n"
29  len = . - msg              # length

```

```

1 #version for 64 bit
2 #worked on testing machine
3 #Linux 4acada6c5b79 4.4.0-93-generic #116~14.04.1-Ubuntu SMP Mon Aug 14 16:07:05 UTC 2017
   x86_64 x86_64 x86_64 GNU/Linux
4
5
6 .global _start
7
8 .text
9
10 _start:
11     # using invalid syscall no 31415 instead of 1 for write()
12     mov     $400, %rax
13     mov     $1, %rdi
14     mov     $message, %rsi
15     mov     $13, %rdx
16     syscall
17
18     # exit(5)
19     mov     $60, %rax
20     mov     $5, %rdi
21     syscall
22     message:
23     .ascii  "Hello, \world\n"

```

2 Experimental Screenshots

```
shiyanolou:HelloWorld/ $ make [2:10:34]
as hello32.s -o hello32.o --32
ld hello32.o -o hello32 -m elf_i386
as hello64.s -o hello64.o --64
ld hello64.o -o hello64 -m elf_x86_64
as helloerr32.s -o helloerr32.o --32
ld helloerr32.o -o helloerr32 -m elf_i386
as helloerr64.s -o helloerr64.o --64
ld helloerr64.o -o helloerr64 -m elf_x86_64
as helloworldExit32.s -o helloworldExit32.o --32
ld helloworldExit32.o -o helloworldExit32 -m elf_i386
as helloworldExit64.s -o helloworldExit64.o --64
ld helloworldExit64.o -o helloworldExit64 -m elf_x86_64
shiyanolou:HelloWorld/ $ ./hello32 [2:10:36]
Hello, world!
shiyanolou:HelloWorld/ $ ./hello64 [2:10:45]
Hello, world
shiyanolou:HelloWorld/ $ ./helloworldExit32 [2:10:48]
Hello, world!
[1] 602 segmentation fault ./helloworldExit32
shiyanolou:HelloWorld/ $ ./helloworldExit64 [2:10:56]
Hello, world
[1] 607 segmentation fault ./helloworldExit64
shiyanolou:HelloWorld/ $ ./helloerr32 [2:11:05]
shiyanolou:HelloWorld/ $ ./helloerr64 [2:11:31]
shiyanolou:HelloWorld/ $ echo 'all of the test results without strace'
all of the test results without strace
shiyanolou:HelloWorld/ $ [2:12:23]
```

Figure 1: Test result for programs


```

shiyanolou:HelloWorld/ $ strace ./hello32 [2:15:50]
execve("./hello32", ["../hello32"], [/* 23 vars */]) = 0
[ Process PID=707 runs in 32 bit mode. ]
write(1, "Hello, world!\n", 14Hello, world!
)
    = 14
_exit(5)
    = ?
+++ exited with 5 +++
shiyanolou:HelloWorld/ $ strace ./hello64 [2:15:55]
execve("./hello64", ["../hello64"], [/* 23 vars */]) = 0
write(1, "Hello, world\n", 13Hello, world
)
    = 13
_exit(5)
    = ?
+++ exited with 5 +++
shiyanolou:HelloWorld/ $ strace ./hellowoExit32 [2:15:59]
execve("./hellowoExit32", ["../hellowoExit32"], [/* 23 vars */]) = 0
[ Process PID=723 runs in 32 bit mode. ]
write(1, "Hello, world!\n", 14Hello, world!
)
    = 14
--- SIGSEGV {si_signo=SIGSEGV, si_code=SI_KERNEL, si_addr=0} ---
+++ killed by SIGSEGV +++
[1] 720 segmentation fault strace ./hellowoExit32
shiyanolou:HelloWorld/ $ strace ./hellowoExit64 [2:16:14]
execve("./hellowoExit64", ["../hellowoExit64"], [/* 23 vars */]) = 0
write(1, "Hello, world\n", 13Hello, world
)
    = 13
--- SIGSEGV {si_signo=SIGSEGV, si_code=SI_KERNEL, si_addr=0} ---
+++ killed by SIGSEGV +++
[1] 728 segmentation fault strace ./hellowoExit64
shiyanolou:HelloWorld/ $ [2:16:18]

```

Figure 2: Test result for straces

```

shiyanolou:HelloWorld/ $ strace ./helloerr32 [2:18:23]
execve("./helloerr32", ["../helloerr32"], [/* 23 vars */]) = 0
[ Process PID=798 runs in 32 bit mode. ]
socket_subcall(0x1, 0x8049096, 0xe, 0, 0, 0) = -1 ENOSYS (Function not imple
mented)
_exit(5)
    = ?
+++ exited with 5 +++
shiyanolou:HelloWorld/ $ strace ./helloerr64 [2:19:26]
execve("./helloerr64", ["../helloerr64"], [/* 23 vars */]) = 0
syscall_400(0x1, 0x4000a6, 0xd, 0, 0, 0) = -1 (errno 38)
_exit(5)
    = ?
+++ exited with 5 +++
shiyanolou:HelloWorld/ $ [2:19:29]

```

Figure 3: Test result for straces

```
shiyanolou:Helloworld/ $ ./hellowoExit32
Hello, world!
[1] 896 segmentation fault ./hellowoExit32
shiyanolou:Helloworld/ $ echo $?
139
shiyanolou:Helloworld/ $ ./hellowoExit64
Hello, world
[1] 905 segmentation fault ./hellowoExit64
shiyanolou:Helloworld/ $ echo $?
139
shiyanolou:Helloworld/ $ █
```

Figure 4: Test result for \$?

3 Narrative

3.1 Problem 3

After the system call, the kernel returns to userspace but there is nothing more in the text region, so the programs runs unknown random memory, or 0s until it hit something invalid. The thing is that Memory always contains some value (even if just zeroes), and the processor will try to execute those bytes as instructions. The bytes might not make sense (illegal opcode) or the instructions themselves may cause a fault. And the garbage may get us into endless loop.