CIA Lab 2 Assignment: Computer Architecture

1.1 Binaries and scripts 1. Find examples of binaries and five different interpreter scripts (Hint: use the file command in “/usr/bin”)?

We can search for only the binary files using whereis -b

I- Binary: xkill

```
kalachkar@desktop-15:/usr/bin$ whereis xkill
xkill: /usr/bin/xkill /usr/share/man/man1/xkill.1.gz
kalachkar@desktop-15:/usr/bin$ file xkill
xkill: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
linked,
interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32,
BuildID[sha1]=dca8e8842cd0151052be4473b45372cd1ae61691, stripped
kalachkar@desktop-15:/usr/bin$ cat xkill
```

II- Binary: tr

```
kalachkar@desktop-15:/usr/bin$ whereis tr
tr: /usr/bin/tr /usr/share/man/man1/tr.1.gz
kalachkar@desktop-15:/usr/bin$ file tr
tr: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32,
BuildID[sha1]=9e38346c47e08de087e9c38df8a7cf1ec67f717f, stripped
kalachkar@desktop-15:/usr/bin$ cat tr
```

II- Using the manual way to search and proof an Interpreter Script: lz

```
kalachkar@desktop-15:/usr/bin$ whereis lz
lz: /usr/bin/lz /usr/share/man/man1/lz.1.gz
kalachkar@desktop-15:/usr/bin$ cat lz | head -2
#!/bin/sh
# Copyright 1994,2002 David C. Niemi.
```

III- Using grep to find all the files that contains “#!/” pattern which means that this file is an interpreter script:

kalachkar@desktop-15:/usr/binLaTeX render failed cat lcf | head -1 #!/bin/bash </code>

1.2 Tracing binaries 2. Use “strace” to find what other system call besides “stat” “zsh” uses before executing an “execve” system call? (Hint: use the “-c” option of “zsh”)

First, I installed zsh on my server. kotaiba@bristol:~ man zsh / for -c option

Take the first argument as a command to execute, rather than reading commands from a script or standard input. If any fur- ther arguments are given, the first one is assigned to strace zsh -c stat

The system call: brk, nmao, open, fstat, read, mprotect, readlink, munmap, lseek, rt_sigprocmask, fcntl.

Sources: 1- http://www.thegeekstuff.com/2011/11/strace-examples/

1.3 Stracing strace 3. (Bonus) Run “strace /bin/pwd” and save the result. Also run “strace strace /bin/pwd” and save the result. How are these outputs related? Explain the “2” in “write(2, ...”.

1.4 ELF format 4. Execute “readelf -Wh «your favorite ELF binary»”. Match the results for the ELF header with the information on ELF’s Wikipedia page. Is this a definitive source for the ELF format? If not, what is?

-Wh option means: W = wide, h = File Header

a- kalachkar@desktop-15:~LaTeX render failed gcc -no-pie -o hello-exe hello.c </code> Now we have LSB executable so ltrace will understand it.

```
kalachkar@desktop-15:~/Desktop$ ltrace ./hello-exe
puts("Give me a break and bring me bac"...Give me a break and bring me back
to life!
)                  = 43
+++ exited (status 225) +++
```

We have only one function calls which is puts().

```
kalachkar@desktop-15:~/Desktop$ ltrace -S ./hello-exe
SYS_brk(0)                                                   = 0xd8c000
SYS_access("/etc/ld.so.nohwcap", 00)                         = -2
SYS_mmap(0, 0x3000, 3, 34)                                   = 0x7f08bd2a4000
SYS_access("/etc/ld.so.preload", 04)                         = -2
SYS_open("/etc/ld.so.cache", 524288, 01)                     = 3
SYS_fstat(3, 0x7ffdb5f33310)                                 = 0
SYS_mmap(0, 0x1f0cf, 1, 2)                                   = 0x7f08bd284000
SYS_close(3)                                                 = 0
SYS_access("/etc/ld.so.nohwcap", 00)                         = -2
SYS_open("/lib/x86_64-linux-gnu/libc.so.6", 524288, 027512510550) = 3
SYS_read(3, "\177ELF\002\001\001\003", 832)                  = 832
SYS_fstat(3, 0x7ffdb5f33350)                                 = 0
SYS_mmap(0, 0x3c69a0, 5, 2050)                               = 0x7f08bccbb000
SYS_mprotect(0x7f08bce79000, 2093056, 0)                     = 0
SYS_mmap(0x7f08bd078000, 0x6000, 3, 2066)                    = 0x7f08bd078000
SYS_mmap(0x7f08bd07e000, 0x39a0, 3, 50)                      = 0x7f08bd07e000
SYS_close(3)                                                 = 0
SYS_mmap(0, 8192, 3, 34)                                     = 0x7f08bd282000
SYS_arch_prctl(4098, 0x7f08bd282700, 0xffff80f742d7d000, 34) = 0
SYS_mprotect(0x7f08bd078000, 16384, 1)                       = 0
SYS_mprotect(0x600000, 4096, 1)                              = 0
SYS_mprotect(0x7f08bd2a7000, 4096, 1)                        = 0
SYS_munmap(0x7f08bd284000, 127183)                           = 0
puts("Give me a break and bring me bac"... <unfinished ...>
SYS_fstat(1, 0x7ffdb5f33ab0)                                 = 0
SYS_brk(0)                                                   = 0xd8c000
SYS_brk(0xdad000)                                            = 0xdad000
SYS_write(1, "Give me a break and bring me bac"..., 43Give me a break and
bring me back to life!
)      = 43
<... puts resumed> )                                            = 43
SYS_exit_group(2017 <no return ...>
+++ exited (status 225) +++
```

As we see the actual exit code is (255), which means exit status out of range because exit takes only integer args in the range 0-255 and here we return '2017'.

Whatever I try to put as return value in main, It gave me a different exit code.

AFTER FEEDBACK CORRECTION

I used:

```
kalachkar@desktop-15:~/Desktop$ bash -c 'exit 2017'; echo $?
225
```

This means that the number is an unsigned byte. When using 255 it works; when using 256 it shows zero.

Sources: 1- http://gcc.gnu.org/onlinedocs/cpp/Preprocessor-Output.html 1- https://www.calleerlandsson.com/the-four-stages-of-compiling-a-c-program/ 3- http://tldp.org/LDP/abs/html/exitcodes.html

3 Inline assembly

12. Create assembly language (for OS3 ASM) for calculating the following formulas, where a, b, c and d are (unsigned) 64-bit integers.

I- a+b In order to have better illustration on the results I put the values of to : rax, rbx, rcx, rdx to 0x00

```
uint64_t rax = 0x00;
uint64_t rbx = 0x00;
uint64_t rcx = 0x00;
uint64_t rdx = 0x00;
```

Now, the addition process for a = 7 and b = 2:

```
#define OS3_ASM \
    "addq $7, %%rax;" \
    "addq $2, %%rbx;" \
    "addq %%rbx, %%rcx;" \
    "addq %%rax, %%rcx;"
```

It gives me this result:

```
kalachkar@desktop-15:~/Desktop$ ./a.out
Before assembly code...
rax: 0000000000000000
```

```
rbx: 0000000000000000
rcx: 0000000000000000
rdx: 0000000000000000

After assembly code...
rax: 0000000000000007
rbx: 0000000000000002
rcx: 0000000000000009
rdx: 0000000000000000
```

II- bc

For multiplication we use imultq:

```
#define OS3_ASM \
    "movq $4, %%rax;" \
    "movq $2, %%rbx;" \
    "movq %%rbx, %%rcx;" \
    "imulq %%rax, %%rcx;"
```

```
kalachkar@desktop-15:~/Desktop$ ./a.out
Before assembly code...
rax: 0000000000000000
rbx: 0000000000000000
rcx: 0000000000000000
rdx: 0000000000000000

After assembly code...
rax: 0000000000000004
rbx: 0000000000000002
rcx: 0000000000000008
rdx: 0000000000000000
```

III- b^2 - 4ac

I assumed b = 4, a = 2, c = 3

I will clarify using line number. 1- put 4 in b 2- Multiply b by itself and store it in b 3- put 2 in a 4- multiply 4 with a and store it in a 5- put 3 in c 6- multiply a with c and store it in c 7- subtract c from b

```
#define OS3_ASM \
    "movq $4, %%rbx;" \
    "imulq %%rbx, %%rbx;" \
    "movq $2, %%rax;" \
    "imulq $4, %%rax;" \
    "movq $3, %%rcx;" \
    "imulq %%rax, %%rcx;" \
    "subq %%rbx, %%rcx;"
```

```
kalachkar@desktop-15:~/Desktop$ ./a.out
Before assembly code...
```

```
rax: 0000000000000000
rbx: 0000000000000000
rcx: 0000000000000000
rdx: 0000000000000000

After assembly code...
rax: 0000000000000008
rbx: 0000000000000010
rcx: 0000000000000008
rdx: 0000000000000000
```

IV- d^4 + d^3 + d^2 + d + 1

I assumed d = 2

I will clarify only first d^4 because the others same process.

Lines 1-4: put 2 in address rax and multiply it by 2 four times so we get same result as 2^4 Notice that I can write it in simpler way, but I have no time. However, if you want me to do that. I WILL. Because this assembly code is really HARD CODED. <code> #define OS3_ASM \ "addq $2, rax;“ \

```
  "imulq $2, %%rax;" \
  "imulq $2, %%rax;" \
  "imulq $2, %%rax;" \
  "addq $2, %%rbx;" \
  "imulq $2, %%rbx;" \
  "imulq $2, %%rbx;" \
  "movq $2, %%rcx;" \
  "imulq $2, %%rcx;" \
  "addq $2, %%rdx;" \
  "addq %%rax, %%rdx;" \
  "addq %%rbx, %%rdx;" \
  "addq %%rcx, %%rdx;" \
  "addq $1, %%rdx;"
```

</code>

Output:

```
kalachkar@desktop-15:~/Desktop$ ./a.out
Before assembly code...
rax: 0000000000000000
rbx: 0000000000000000
rcx: 0000000000000000
rdx: 0000000000000000

After assembly code...
rax: 0000000000000010
rbx: 0000000000000008
rcx: 0000000000000004
rdx: 000000000000001f
```

```
<code>

Hex to number:
0000000000000010 = 16
0000000000000008 = 8
0000000000000004 = 4
000000000000001f = 31

lets write it assuming d = 2:

2^4 + 2^3 + 2^2 + 2 + 1 = 1


//Sources://
1- https://en.wikibooks.org/wiki/X86_Assembly/GAS_Syntax
2- https://www3.nd.edu/~dthain/courses/cse40243/fall2015/intel-intro.html



14. Compile the program above with the “-g” flag to generate debugging
information.
Run the program. What is the purpose of these bytes (15, 162)?

the bytes .byte 15 and .byte 162 represent the CPUID instruction.


//Source://
1-
https://stackoverflow.com/questions/35230027/what-does-byte-mean-in-this-asm
-line/35230349#35230349


15. Why does the program exit with 052?

GDB:
<code>
Starting program: /sne/home/kalchkar/Desktop/inspect
GenuineIntel
[Inferior 1 (process 4957) exited with code 052]
<code>

In terminal:
<code>
kalachkar@desktop-15:~/Desktop$ ./inspect
GenuineIntel
kalachkar@desktop-15:~/Desktop$ echo $?
42
```

After asking my colleague Tim the exit-code of gdb is in octal format. If you convert 52 from octal to decimal it will give you 42 . Which means that when we used the exit code of last command we used in the terminal it gives the exit-code in decimals and gdb in octal format. So 053 is the same as 42.