

Task 1 Printing on the Screen [30%]

(a) In `print_stk.asm`, explain how the line “`push 0x000a0d21`” works. Show a screenshot from `gdb` to support your explanation.

- (1) This push instruction first decrements esp by 4 (from 0xbffff72c to 0xbffff728)
- (2) Then it places its operand "0x000a0d21" into the contents of the 32-bit location at address [esp] (0xbffff728)

Actually, the single instruction "push 0x000a0d21" is equivalent to these two instructions:

```
sub esp, 4
```

```
mov dword [esp], 0x000a0d21
```

A screenshot before executing the instruction "push 0x000a0d21" :

```

[ Legend: Modified register | Code | Heap | Stack | String ]

registers
eax    : 0x0
ecx    : 0x0
edx    : 0x0
esp    : 0xbffff72c → 0x00000000
ebp    : 0x0
esi    : 0x0
edi    : 0x0
eip    : 0x00400666 → "hll\r\n"
eflags: [carry parity adjust zero sign trap INTERRUPT direction overflow RESUME virtualx86 identification]
cs: 0x0073  eip: 0x007b  ds: 0x007b  es: 0x007b  fs: 0x0000  gs: 0x0000

stack
0xbffff72c: +0x0000: 0x00000000 ← $esp
0xbffff730: +0x0004: 0x00000001
0xbffff734: +0x0008: 0xbffff863 → "/home/kalyu/Lab02/print_stk"
0xbffff738: +0x000c: 0x00000000
0xbffff73c: +0x0010: 0xbffff87f → "TERM=xterm-256color"
0xbffff740: +0x0014: 0xbffff893 → "SHELL=/bin/bash"
0xbffff744: +0x0018: 0xbffff8a3 → "USER=root"
0xbffff748: +0x001c: 0xbffff8ad → "LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so[...]"

code:x86:32
0x804805d: add     BYTE PTR [eax], al
0x804805f: add     BYTE PTR [eax+0x0], bh
0x8048065: <_start+5>    push    eax
→ 0x8048066: <_start+6>    push    0xa0d21
0x804806b: <_start+11>   push    0x646726f
0x8048070: <_start+16>   push    0x77202c6f
0x8048075: <_start+21>   push    0x6c6c6548
0x804807a: <_start+26>   mov     eax, 0x4
0x804807f: <_start+31>   mov     ebx, 0x1

threads
[#0] Id 1, Name: "print_stk", stopped 0x8048066 in _start (), reason: SINGLE STEP

trace
[#0] 0x8048066 → _start()

```

A screenshot after executing the instruction "push 0x000a0d21" :

```

[ Legend: Modified register | Code | Heap | Stack | String ]

registers
eax    : 0x0
ecx    : 0x0
edx    : 0x0
esp    : 0xbffff728 → "!\\r\\n"
ebp    : 0x0
esi    : 0x0
edi    : 0x0
eip    : 0x0048066 → <_start+11> push 0x646c726f
eflags: [carry parity adjust zero sign trap INTERRUPT direction overflow RESUME virtualx86 identification]
cs: 0x073  eip: 0x07b  ds: 0x07b  es: 0x07b  fs: 0x000  gs: 0x000

stack
0xbffff728+0x0000: "!\\r\\n" ← $esp
0xbffff72c+0x0004: 0x00000000
0xbffff730+0x0008: 0x00000001
0xbffff734+0x000c: 0xbffff728 → "/home/kalyu/Lab02/print_stk"
0xbffff738+0x0010: 0x00000000
0xbffff73c+0x0014: 0xbffff7ff → "TERM=xterm-256color"
0xbffff740+0x0018: 0xbffff893 → "SHELL=/bin/bash"
0xbffff744+0x001c: 0xbffff8a3 → "USER=root"

code:x86:32
0x004805ff          add     BYTE PTR [eax+0x0], bh
0x00480665 <_start+5>    push   eax
0x00480666 <_start+6>    push   0xa0d21
→ 0x0048066b <_start+11> push   0x646c726f
0x00480670 <_start+16>    push   0x77202c5f
0x00480675 <_start+21>    push   0x6c6c6548
0x0048067a <_start+26>    mov     eax, 0x4
0x0048067f <_start+31>    mov     ebx, 0x1
0x00480684 <_start+36>    mov     ecx, esp

threads
[#0] Id 1, Name: "print_stk", stopped 0x0048066b in _start(), reason: SINGLE STEP

trace
[#0] 0x0048066b → _start()

```

(b) Also, in the same file, explain how you got the string address. Show a screenshot from gdb to support your explanation.

Because the string "Hello, world!\r\n" is stored on the stack, we can get the string address by copying the value of the **esp** register into the **ecx** register.

Below is the screenshot before executing the instruction "mov ecx, esp".

We can see that now the **esp** register points the string "Hello, world!\r\n", which means the **esp** register now holds the address of the string.



```
[ Legend: Modified register | Code | Heap | Stack | String ]

registers
eax : 0x4
ebx : 0x1
ecx : 0x0
edx : 0x0
esp : 0xbffff71c → "Hello, world!\r\n"
ebp : 0x0
esi : 0x0
edi : 0x0
eip : 0x8048084 → <_start+36> mov ecx, esp
eflags: [carry parity adjust zero sign trap INTERRUPT direction overflow RESUME virtualx86 identification]
cs: 0x0073  eip: 0x007b  ds: 0x007b  es: 0x007b  fs: 0x0000  gs: 0x0000

stack
0xbffff71c +0x0000: "Hello, world!\r\n" ← $esp
0xbffff720 +0x0004: "o, world!\r\n"
0xbffff724 +0x0008: "orld!\r\n"
0xbffff728 +0x000c: "\r\n"
0xbffff72c +0x0010: 0x00000000
0xbffff730 +0x0014: 0x00000001
0xbffff734 +0x0018: 0xbffff000 → "/home/kaiyu/Lab02/print_stk"
0xbffff738 +0x001c: 0x00000000

code:x86:32
0x8048075 <_start+21> push 0x6c6c6548
0x804807a <_start+26> mov eax, 0x4
0x804807f <_start+31> mov ebx, 0x1
→ 0x8048084 <_start+36> mov ecx, esp
0x8048086 <_start+38> mov edx, 0xf
0x804808b <_start+43> int 0x80
0x804808d <_start+45> mov eax, 0x1
0x8048092 <_start+50> mov ebx, 0x0
0x8048097 <_start+55> int 0x80

threads
[#0] Id 1, Name: "print_stk", stopped 0x8048084 in _start (), reason: SINGLE STEP

trace
[#0] 0x8048084 → _start()
```

So, we can copy the address the value of the **esp** register into the **ecx** register to get the address of the string "Hello, world!\r\n".

And this is the screenshot after executing the instruction "mov ecx, esp":



```
[ Legend: Modified register | Code | Heap | Stack | String ]

registers
eax : 0x4
ebx : 0x1
→ ecx : 0xbffff71c → "Hello, world!\r\n"
edx : 0x0
esp : 0xbffff71c → "Hello, world!\r\n"
ebp : 0x0
esi : 0x0
edi : 0x0
eip : 0x8048086 → <_start+38> mov edx, 0xf
eflags: [carry parity adjust zero sign trap INTERRUPT direction overflow RESUME virtualx86 identification]
cs: 0x0073  eip: 0x007b  ds: 0x007b  es: 0x007b  fs: 0x0000  gs: 0x0000

stack
0xbffff71c +0x0000: "Hello, world!\r\n" ← $esp
0xbffff720 +0x0004: "o, world!\r\n"
0xbffff724 +0x0008: "orld!\r\n"
0xbffff728 +0x000c: "\r\n"
0xbffff72c +0x0010: 0x00000000
0xbffff730 +0x0014: 0x00000001
0xbffff734 +0x0018: 0xbffff000 → "/home/kaiyu/Lab02/print_stk"
0xbffff738 +0x001c: 0x00000000

code:x86:32
0x804807a <_start+26> mov eax, 0x4
0x804807f <_start+31> mov ebx, 0x1
0x8048084 <_start+36> mov ecx, esp
→ 0x8048086 <_start+38> mov edx, 0xf
0x804808b <_start+43> int 0x80
0x804808d <_start+45> mov eax, 0x1
0x8048092 <_start+50> mov ebx, 0x0
0x8048097 <_start+55> int 0x80
0x8048099 add BYTE PTR [eax], al

threads
[#0] Id 1, Name: "print_stk", stopped 0x8048086 in _start (), reason: SINGLE STEP

trace
[#0] 0x8048086 → _start()
```

Task 2 Spawning a Shell [70%]

Startup Code (labsh.asm) [10%]

The process number of the **calling shell** is **2261**, and the process number of the **spawned shell** is **2456**.

The passed environment variable to the spawned shell is "**PWD=/home/kaiyu/Lab02**".

Below is a screenshot of a successful run:

```

root@kaiyu:/home/kaiyu/Lab02# nasm -f elf labsh.s
root@kaiyu:/home/kaiyu/Lab02# ld -o labsh labsh.o
root@kaiyu:/home/kaiyu/Lab02# ./labsh
# echo $$
2456
# /usr/bin/env
PWD=/home/kaiyu/Lab02
# exit
root@kaiyu:/home/kaiyu/Lab02# echo $$
2261
root@kaiyu:/home/kaiyu/Lab02# █

```

Providing Arguments to /bin/sh [20%]

Below is a screenshot of a successful run:

```

root@kaiyu:/home/kaiyu/Lab02# ./labsh_args
total 124
drwxr-xr-x  2 root  root  4096 Jan 15 21:58 .
drwxr-xr-x 19 kaiyu kaiyu 4096 Jan 16 13:40 ..
-rw-r--r--  1 root  root   289 Jan 12 14:42 code.c
-rwxr-xr-x  1 root  root   648 Jan 12 14:36 helloworld
-rw-r--r--  1 root  root   624 Jan 12 14:36 helloworld.o
-rw-r--r--  1 root  root   388 Jan 12 14:35 helloworld.s
-rwxr-xr-x  1 root  root   512 Jan 13 17:14 labsh
-rw-r--r--  1 root  root   448 Jan 13 17:14 labsh.o
-rw-r--r--  1 root  root   630 Jan 13 16:53 labsh.s
-rwxr-xr-x  1 root  root   548 Jan 15 13:27 labsh_args
-rw-r--r--  1 root  root   496 Jan 15 13:27 labsh_args.o
-rw-r--r--  1 root  root   896 Jan 15 13:27 labsh_args.s
-rwxr-xr-x  1 root  root   576 Jan 15 16:25 labsh_env
-rw-r--r--  1 root  root   512 Jan 15 16:25 labsh_env.o
-rw-r--r--  1 root  root  1097 Jan 15 16:25 labsh_env.s
-rwxr-xr-x  1 root  root   576 Jan 15 19:51 labsh_rel
-rw-r--r--  1 root  root   512 Jan 15 19:51 labsh_rel.o
-rw-r--r--  1 root  root   545 Jan 15 19:51 labsh_rel.s
-rwxr-xr-x  1 root  root   488 Jan 12 10:48 mini
-rw-r--r--  1 root  root   432 Jan 12 10:48 mini.o
-rw-r--r--  1 root  root    77 Jan 12 10:45 mini.text
-rwxr-xr-x  1 root  root  7344 Jan 12 14:40 mini_shelltest
-rwxr-xr-x  1 root  root   604 Jan 12 15:17 print_rel
-rw-r--r--  1 root  root   544 Jan 12 15:17 print_rel.o
-rw-r--r--  1 root  root   610 Jan 15 12:49 print_rel.s
-rwxr-xr-x  1 root  root   540 Jan 15 12:49 print_stk
-rw-r--r--  1 root  root   480 Jan 15 12:49 print_stk.o
-rw-r--r--  1 root  root   589 Jan 15 12:49 print_stk.s
-rwxr-xr-x  1 root  root  7404 Jan 12 14:43 shelltest

```

A screenshot of running the command "ls -la" in the same path:

```

root@kaiyu:/home/kaiyu/Lab02# ls -la
total 124
drwxr-xr-x  2 root  root  4096 Jan 15 21:58 .
drwxr-xr-x 19 kaiyu kaiyu 4096 Jan 16 13:40 ..
-rw-r--r--  1 root  root   289 Jan 12 14:42 code.c
-rwxr-xr-x  1 root  root   648 Jan 12 14:36 helloworld
-rw-r--r--  1 root  root   624 Jan 12 14:36 helloworld.o
-rw-r--r--  1 root  root   388 Jan 12 14:35 helloworld.s
-rwxr-xr-x  1 root  root   512 Jan 13 17:14 labsh
-rwxr-xr-x  1 root  root   548 Jan 15 13:27 labsh_args
-rw-r--r--  1 root  root   496 Jan 15 13:27 labsh_args.o
-rw-r--r--  1 root  root   896 Jan 15 13:27 labsh_args.s
-rwxr-xr-x  1 root  root   576 Jan 15 16:25 labsh_env
-rw-r--r--  1 root  root   512 Jan 15 16:25 labsh_env.o
-rw-r--r--  1 root  root  1097 Jan 15 16:25 labsh_env.s
-rw-r--r--  1 root  root   448 Jan 13 17:14 labsh.o
-rwxr-xr-x  1 root  root   576 Jan 15 19:51 labsh_rel
-rw-r--r--  1 root  root   512 Jan 15 19:51 labsh_rel.o
-rw-r--r--  1 root  root   545 Jan 15 19:51 labsh_rel.s
-rw-r--r--  1 root  root   630 Jan 13 16:53 labsh.s
-rwxr-xr-x  1 root  root   488 Jan 12 10:48 mini
-rw-r--r--  1 root  root   432 Jan 12 10:48 mini.o
-rwxr-xr-x  1 root  root  7344 Jan 12 14:40 mini_shelltest
-rw-r--r--  1 root  root    77 Jan 12 10:45 mini.text
-rwxr-xr-x  1 root  root   604 Jan 12 15:17 print_rel
-rw-r--r--  1 root  root   544 Jan 12 15:17 print_rel.o
-rw-r--r--  1 root  root   610 Jan 15 12:49 print_rel.s
-rwxr-xr-x  1 root  root   540 Jan 15 12:49 print_stk
-rw-r--r--  1 root  root   480 Jan 15 12:49 print_stk.o
-rw-r--r--  1 root  root   589 Jan 15 12:49 print_stk.s
-rwxr-xr-x  1 root  root  7404 Jan 12 14:43 shelltest

```

Providing Env. Variables to /bin/sh [20%]

Below is a screenshot of a successful run:

```

root@kaiyu:/home/kaiyu/Lab02# vim labsh_env.s
root@kaiyu:/home/kaiyu/Lab02# nasm -f elf labsh_env.s
root@kaiyu:/home/kaiyu/Lab02# ld -o labsh_env labsh_env.o
root@kaiyu:/home/kaiyu/Lab02# ./labsh_env
# /usr/bin/env
bbbb=5678
aaaa=1234
cccc=1234
PWD=/home/kaiyu/Lab02
#

```

Using the Relative Addressing Technique [20%]

To set the text and data sections to be readable and writable, we should use the command:

`ld --omagic labsh_rel.o -o labsh_rel,`

If we didn't add the `--omagic` option, it will report a segmentation fault.

Below is a screenshot of a successful run:

```

root@kaiyu:/home/kaiyu/Lab02# nasm -f elf labsh_rel.s
root@kaiyu:/home/kaiyu/Lab02# ld --omagic labsh_rel.o -o labsh_rel
root@kaiyu:/home/kaiyu/Lab02# ./labsh_rel
# echo $$
3576
# exit
root@kaiyu:/home/kaiyu/Lab02# echo $$
2246
root@kaiyu:/home/kaiyu/Lab02# █

```

A detailed explanation for each line of the code: ()

```

section .text
global _start
_start:
    jmp two
one:
    pop esi
    ; esi now holds address of the string "/bin/sh*AAAABBBB"
    mov ebx, esi
    ; (complete) ebx should contain the string address
    mov eax, 0
    mov byte [ebx+7], 0x00
    ; (complete) terminate /bin/sh with 0x00 (1 byte)
    ; ebx now holds the address of the string "/bin/sh"

    ; start to construct the array argv[]
    mov [ebx+8], ebx
    ; (complete) save ebx to memory at address ebx+8
    ; replace AAAA with the address of the string "/bin/sh"
    mov [ebx+12], eax
    ; (complete) save eax to memory at address ebx+12
    ; replace BBBB with NULL bytes
    lea ecx, [ebx+8]
    ; let ecx = ebx + 8
    ; now ecx points to the array argv[]
    ; argv[0] = NULL
    ; argv[1] = the address of string "/bin/sh"

    ; For environment variable
    mov edx, 0
    ; No env variables

    ; Call execve()
    mov al, 0x0b
    ; eax = 0x0000000b
    int 0x80
two:

```

```

call one
; now the string '/bin/sh*AAAABBBB' is stored on the stack
db '/bin/sh*AAAABBBB'

```

Why this code would successfully execute the `/bin/sh` program?

How the `argv` array is constructed?

(1) We know that the system call interface: `sys_execve()` is:

```

asmlinkage long sys_execve(const char __user *filename,
                           const char __user *const __user *argv,
                           const char __user *const __user *envp);

```

The first parameter `filename` should be stored in `ebx` register, the second parameter `argv` should be stored in `ecx` register, and the third parameter `envp` should be stored in `edx` register.

(2) To execute the `/bin/sh` program, we should move address of string `"/bin/sh0"` into `ebx`, move address of the address of `"/bin/sh0"` into `ecx`, and move address of `NULL` word into `edx`.

So, we need the null terminated string `/bin/sh` somewhere in memory, and we need the address of the string `/bin/sh` somewhere in memory, and we need we need a `NULL` word somewhere in memory.

(3) By using the Relative Addressing Technique, we first use `jmp` instruction at beginning of shellcode to jump to the `call` instruction, which is right before the string `'/bin/sh*AAAABBBB'`.

Then we use `call` instruction to jump back to first instruction after `jmp`.

Now the address of the string `'/bin/sh*AAAABBBB'` is on the stack.

Legend: Modified register | Code | Heap | Stack | String

Registers:

- \$eax : 0x0
- \$ebx : 0x0
- \$ecx : 0x0
- \$edx : 0x0
- \$esp : 0xbffff71c → 0x00048085 → **"/bin/sh*AAAABBBB"**
- \$ebp : 0x0
- \$esi : 0x0
- \$edi : 0x0
- \$eip : 0x00048062 → <one+0> pop esi

SeFlags: [carry parity adjust zero sign trap INTERRUPT direction overflow RESUME virtualx86 identification]
 \$cs: 0x0073 \$ss: 0x007b \$ds: 0x007b \$es: 0x007b \$fs: 0x0000 \$gs: 0x0000

Stack:

- 0xbffff71c | 0x0000: 0x00048085 → **"/bin/sh*AAAABBBB"** ← \$esp
- 0xbffff728 | 0x0004: 0x00000001
- 0xbffff724 | 0x0008: 0xbffff859 → **"/home/kalyu/Lab02/Labsh_rel"**
- 0xbffff728 | 0x000c: 0x00000000
- 0xbffff72c | 0x0010: 0xbffff875 → **"TERM=xterm-256color"**
- 0xbffff730 | 0x0014: 0xbffff889 → **"SHELL=/bin/bash"**
- 0xbffff734 | 0x0018: 0xbffff899 → **"USER=root"**
- 0xbffff738 | 0x001c: 0xbffff8a3 → **"LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so[...]"**

Code (x86:32):

- 0x804805d | add BYTE PTR [eax], al
- 0x804805f | add bl, ch
- 0x8048061 | <_start+1> push ds
- 0x8048062 | <one+0> pop esi
- 0x8048063 | <one+1> mov ebx, esi
- 0x8048065 | <one+3> mov eax, 0x0
- 0x804806a | <one+8> mov BYTE PTR [ebx+0x7], 0x0
- 0x804806e | <one+12> mov DWORD PTR [ebx+0x8], ebx
- 0x8048071 | <one+15> mov DWORD PTR [ebx+0xc], eax

(4) Then we use the instruction `"pop esi"`, now `esi` holds the address of the string `'/bin/sh*AAAABBBB'`.

```

[ Legend: Modified register | Code | Heap | Stack | String ]
registers
$eax : 0x0
$ebx : 0x0
$ecx : 0x0
$edx : 0x0
$esp : 0xbffff720 → 0x00000001
$esi : 0x00040005 → "/bin/sh*AAAABBBB"
$edi : 0x0
$ebp : 0xbfffa0063 → <one+1> mov ebx, esi
$eflags: [carry parity adjust zero sign trap INTERRUPT direction overflow RESUME virtualx86 identification]
$cs: 0x0073 $ss: 0x007b $ds: 0x007b $es: 0x007b $fs: 0x0000 $gs: 0x0000

stack
0xbffff720 +0x0000: 0x00000001 ← $esp
0xbffff724 +0x0004: 0xbffff859 → "/home/kaiyu/Lab02/labsh_rel"
0xbffff728 +0x0008: 0x00000000 → "TERM=xterm-256color"
0xbffff72c +0x000c: 0xbffff875 → "SHELL=/bin/bash"
0xbffff730 +0x0010: 0xbffff889 → "USER=root"
0xbffff734 +0x0014: 0xbffff899 → "USER=root"
0xbffff738 +0x0018: 0xbffff8a3 → "LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so[...]"
0xbffff73c +0x001c: 0xbffff82b → "SUDO_USER=kaiyu"

code:x86:32
0x804805f add bl, ch
0x8048061 <_start+1> push ds
0x8048062 <one+0> pop esi
→ 0x8048063 <one+1> mov ebx, esi
0x8048065 <one+3> mov eax, 0x0
0x804806a <one+8> mov BYTE PTR [ebx+0x7], 0x0
0x804806e <one+12> mov DWORD PTR [ebx+0x8], ebx
0x8048071 <one+15> mov DWORD PTR [ebx+0xc], eax
0x8048074 <one+18> lea ecx, [ebx+0x8]

```

(5) Then we use instruction "mov ebx, esi", now ebx holds the address of the string '/bin/sh*AAAABBBB'.

```

[ Legend: Modified register | Code | Heap | Stack | String ]
registers
$eax : 0x0
$ebx : 0x00040005 → "/bin/sh*AAAABBBB"
$ecx : 0x0
$edx : 0x0
$esp : 0xbffff720 → 0x00000001
$ebp : 0x0
$esi : 0x00040005 → "/bin/sh*AAAABBBB"
$edi : 0x0
$ebp : 0xbfffa0065 → <one+3> mov eax, 0x0
$eflags: [carry parity adjust zero sign trap INTERRUPT direction overflow RESUME virtualx86 identification]
$cs: 0x0073 $ss: 0x007b $ds: 0x007b $es: 0x007b $fs: 0x0000 $gs: 0x0000

stack
0xbffff720 +0x0000: 0x00000001 ← $esp
0xbffff724 +0x0004: 0xbffff859 → "/home/kaiyu/Lab02/labsh_rel"
0xbffff728 +0x0008: 0x00000000 → "TERM=xterm-256color"
0xbffff72c +0x000c: 0xbffff875 → "SHELL=/bin/bash"
0xbffff730 +0x0010: 0xbffff889 → "USER=root"
0xbffff734 +0x0014: 0xbffff899 → "USER=root"
0xbffff738 +0x0018: 0xbffff8a3 → "LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so[...]"
0xbffff73c +0x001c: 0xbffff82b → "SUDO_USER=kaiyu"

code:x86:32
0x8048061 <_start+1> push ds
0x8048062 <one+0> pop esi
0x8048063 <one+1> mov ebx, esi
→ 0x8048065 <one+3> mov eax, 0x0
0x804806a <one+8> mov BYTE PTR [ebx+0x7], 0x0
0x804806e <one+12> mov DWORD PTR [ebx+0x8], ebx
0x8048071 <one+15> mov DWORD PTR [ebx+0xc], eax
0x8048074 <one+18> lea ecx, [ebx+0x8]
0x8048077 <one+21> mov edx, 0x0

```

(6) Then we use the instruction "mov byte [ebx+7], 0x00" to replace the "*" of the string "/bin/sh*AAAABBBB" with a zero byte. Now ebx holds the address of the string "/bin/sh".

```

[ Legend: Modified register | Code | Heap | Stack | String ]
registers
$eax : 0x0
$ebx : 0x00040005 → "/bin/sh"
$ecx : 0x0
$edx : 0x0
$esp : 0xbffff720 → 0x00000001
$ebp : 0x0
$esi : 0x00040005 → "/bin/sh"
$edi : 0x0
$ebp : 0xbfffa006e → <one+12> mov DWORD PTR [ebx+0x8], ebx
$eflags: [carry parity adjust zero sign trap INTERRUPT direction overflow RESUME virtualx86 identification]
$cs: 0x0073 $ss: 0x007b $ds: 0x007b $es: 0x007b $fs: 0x0000 $gs: 0x0000

stack
0xbffff720 +0x0000: 0x00000001 ← $esp
0xbffff724 +0x0004: 0xbffff859 → "/home/kaiyu/Lab02/labsh_rel"
0xbffff728 +0x0008: 0x00000000 → "TERM=xterm-256color"
0xbffff72c +0x000c: 0xbffff875 → "SHELL=/bin/bash"
0xbffff730 +0x0010: 0xbffff889 → "USER=root"
0xbffff734 +0x0014: 0xbffff899 → "USER=root"
0xbffff738 +0x0018: 0xbffff8a3 → "LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so[...]"
0xbffff73c +0x001c: 0xbffff82b → "SUDO_USER=kaiyu"

code:x86:32
0x8048063 <one+1> mov ebx, esi
0x8048065 <one+3> mov eax, 0x0
0x804806a <one+8> mov BYTE PTR [ebx+0x7], 0x0
→ 0x804806e <one+12> mov DWORD PTR [ebx+0x8], ebx
0x8048071 <one+15> mov DWORD PTR [ebx+0xc], eax
0x8048074 <one+18> lea ecx, [ebx+0x8]
0x8048077 <one+21> mov edx, 0x0
0x804807c <one+26> mov al, 0xb
0x804807e <one+28> int 0x80

```

The whole string (include zero byte at the middle) in the memory is "/bin/sh0AAAABBBB".


```

[ Legend: Modified register | Code | Heap | Stack | String ]
registers
$eax : 0x0
$ebx : 0x00040085 → "/bin/sh"
$ecx : 0x0
$edx : 0x0
$esp : 0xbffff720 → 0x00000001
$ebp : 0x0
$esi : 0x00040085 → "/bin/sh"
$edi : 0x0
$eip : 0x00040066 → <one+12> mov DWORD PTR [ebx+0x8], ebx
SerFlags: [carry parity adjust zero sign trap INTERRUPT direction overflow RESUME virtualx86 identification]
$cs: 0x0073 $ss: 0x007b $ds: 0x007b $es: 0x007b $fs: 0x0000 $gs: 0x0000

stack
0xbffff720|+0x0000: 0x00000001 ← $esp
0xbffff724|+0x0004: 0xbffff859 → "/home/kaiyu/Lab02/labsh_rel"
0xbffff728|+0x0008: 0x00000000
0xbffff72c|+0x000c: 0xbffff875 → "TERM=xterm-256color"
0xbffff730|+0x0010: 0xbffff889 → "SHELL=/bin/bash"
0xbffff734|+0x0014: 0xbffff899 → "USER=root"
0xbffff738|+0x0018: 0xbffff8a3 → "LS_COLORS=rs=0:di=01;34;ln=01;36:mh=00:pi=40;33:so[...]"
0xbffff73c|+0x001c: 0xbffff82b → "SUDO_USER=kaiyu"

code:x86:32
0x00040063 <one+1> mov ebx, esi
0x00040065 <one+3> mov eax, 0x0
0x0004006a <one+8> mov BYTE PTR [ebx+0x7], 0x0
→ 0x0004006e <one+12> mov DWORD PTR [ebx+0x8], ebx
0x00040071 <one+15> mov DWORD PTR [ebx+0xc], eax
0x00040074 <one+18> lea ecx, [ebx+0x8]
0x00040077 <one+21> mov edx, 0x0
0x0004007c <one+26> mov al, 0xb
0x0004007e <one+28> int 0x80

threads
[#0] Id 1, Name: "labsh_rel", stopped 0x0004006e in one (), reason: SINGLE STEP

trace
[gef] 0x0004006e → one()
gef> x/4 0x0004008d
0x0004008d <two+13>: 0x41414141 0x42424242 0x0 0x0

```

- (7) Then we start to construct the array argv[]. We use the instruction "mov [ebx+8], ebx" to replace "AAAA" of the string "/bin/sh0AAAABBBB" with the address of string "/bin/sh". The whole string (include zero byte at the middle) in the memory is "/bin/sh0'addr'BBBB".

```

[ Legend: Modified register | Code | Heap | Stack | String ]
registers
$eax : 0x0
$ebx : 0x00040085 → "/bin/sh"
$ecx : 0x0
$edx : 0x0
$esp : 0xbffff720 → 0x00000001
$ebp : 0x0
$esi : 0x00040085 → "/bin/sh"
$edi : 0x0
$eip : 0x00040071 → <one+15> mov DWORD PTR [ebx+0xc], eax
SerFlags: [carry parity adjust zero sign trap INTERRUPT direction overflow RESUME virtualx86 identification]
$cs: 0x0073 $ss: 0x007b $ds: 0x007b $es: 0x007b $fs: 0x0000 $gs: 0x0000

stack
0xbffff720|+0x0000: 0x00000001 ← $esp
0xbffff724|+0x0004: 0xbffff859 → "/home/kaiyu/Lab02/labsh_rel"
0xbffff728|+0x0008: 0x00000000
0xbffff72c|+0x000c: 0xbffff875 → "TERM=xterm-256color"
0xbffff730|+0x0010: 0xbffff889 → "SHELL=/bin/bash"
0xbffff734|+0x0014: 0xbffff899 → "USER=root"
0xbffff738|+0x0018: 0xbffff8a3 → "LS_COLORS=rs=0:di=01;34;ln=01;36:mh=00:pi=40;33:so[...]"
0xbffff73c|+0x001c: 0xbffff82b → "SUDO_USER=kaiyu"

code:x86:32
0x00040065 <one+3> mov eax, 0x0
0x0004006a <one+8> mov BYTE PTR [ebx+0x7], 0x0
0x0004006e <one+12> mov DWORD PTR [ebx+0x8], ebx
→ 0x00040071 <one+15> mov DWORD PTR [ebx+0xc], eax
0x00040074 <one+18> lea ecx, [ebx+0x8]
0x00040077 <one+21> mov edx, 0x0
0x0004007c <one+26> mov al, 0xb
0x0004007e <one+28> int 0x80
0x00040080 <two+0> call 0x00040062 <one>

threads
[#0] Id 1, Name: "labsh_rel", stopped 0x00040071 in one (), reason: SINGLE STEP

trace
[gef] 0x00040071 → one()
gef> x/4 0x0004008d
0x0004008d <two+13>: 0x00040085 0x42424242 0x0 0x0

```

There is a mistake in the page 6 of "lab02_explainer.pdf":

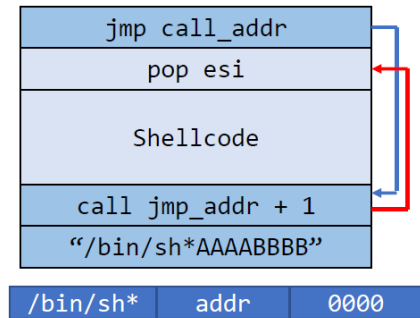
Activity 2: Spawn a new Shell

- A startup code is provided that uses relative addressing, you need to:

- Complete the missing parts
- Answer few questions

- You need to replace:

- * with a NULL byte
- AAAA with the address of the address of string
- BBBB with NULL bytes
- Why cannot we start with `/bin/sh0AAAA0000`?



- Can a program modify the code segment?
 - How can we solve this issue?

We need to replace "AAAA" with the address of string `/bin/sh` instead of the address of address of string `/bin/sh`.

- Then we use the instruction `mov [ebx+12], eax` to replace "BBBB" of the string `/bin/sh0'addr'BBBB` with NULL bytes.

Now we finished constructing the `argv[]`. The address of the array `argv[]` is `ebx+8`.

`argv[0] = NULL`

`argv[1] = the address of string "/bin/sh"`

```
[ Legend: Modified register | Code | Heap | Stack | String ]

Registers
$eax : 0x0
$ebx : 0x00408085 → "/bin/sh"
$ecx : 0x0
$edx : 0x0
$esp : 0xbffff720 → 0x00000001
$ebp : 0x0
$esi : 0x00408085 → "/bin/sh"
$edi : 0x0
$ebp : 0x00408074 → <one+18> lea ecx, [ebx+0x8]
$eflags: [carry parity adjust zero sign trap INTERRUPT direction overflow RESUME virtualx86 identification]
$cs: 0x0073 $ss: 0x007b $ds: 0x007b $es: 0x0000 $fs: 0x0000 $gs: 0x0000

Stack
0xbffff720: 0x00000001 ← $esp
0xbffff724: 0xbffff830 → "/home/kaiyu/Lab02/labsh_rel"
0xbffff728: 0x00000000
0xbffff72c: 0xbffff875 → "TERM=xterm-256color"
0xbffff730: 0xbffff889 → "SHELL=/bin/bash"
0xbffff734: 0xbffff899 → "USER=root"
0xbffff738: 0xbffff8a3 → "LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so[...]"
0xbffff73c: 0xbffff82b → "SUDO_USER=kaiyu"

Code:x86:32
0x804806a <one+8> mov BYTE PTR [ebx+0x7], 0x0
0x804806e <one+12> mov DWORD PTR [ebx+0x8], ebx
0x8048071 <one+15> mov DWORD PTR [ebx+0xc], eax
→ 0x8048074 <one+18> lea ecx, [ebx+0x8]
0x8048077 <one+21> mov edx, 0x0
0x804807c <one+26> mov al, 0xb
0x804807e <one+28> int 0x80
0x8048080 <two+0> call 0x8048062 <one>
0x8048085 <two+5> das

[ #0 ] Id 1, Name: "labsh_rel", stopped 0x8048074 in one (), reason: SINGLE STEP
[ #0 ] 0x8048074 → one()

gef> x/4 0x0040808d
0x0040808d <two+13>: 0x8048085 0x0 0x0 0x0
```

- Then we use the instruction `lea ecx, [ebx+8]` to store the address of the array `argv[]` in the `ecx` register. Now `ecx` points to the array `argv[]`.

```

[ Legend: Modified register | Code | Heap | Stack | String ]
registers
$eax : 0x0
$ebx : 0x00048085 → "/bin/sh"
$ecx : 0x0004808d → 0x00048085 → "/bin/sh"
$edx : 0x0
$esp : 0xbffff720 → 0x00000001
$ebp : 0x0
$esi : 0x00048085 → "/bin/sh"
$edi : 0x0
$ebp : 0x00048077 → <one+21> mov edx, 0x0
$eflags: [carry parity adjust zero sign trap INTERRUPT direction overflow RESUME virtualx86 identification]
$cs: 0x0073 $ss: 0x007b $ds: 0x007b $es: 0x007b $fs: 0x0000 $gs: 0x0000

stack
0xbffff720 +0x0000: 0x00000001 ← $esp
0xbffff724 +0x0004: 0xbffff859 → "/home/kaiyu/Lab02/labsh_rel"
0xbffff728 +0x0008: 0x00000000
0xbffff72c +0x000c: 0xbffff875 → "TERM=xterm-256color"
0xbffff730 +0x0010: 0xbffff889 → "SHELL=/bin/bash"
0xbffff734 +0x0014: 0xbffff899 → "USER=root"
0xbffff738 +0x0018: 0xbffff8a3 → "LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so(...)"
0xbffff73c +0x001c: 0xbffffe2b → "SUDO_USER=kaiyu"

code:x86:32
0x804806e <one+12> mov     DWORD PTR [ebx+0x8], ebx
0x8048071 <one+15> mov     DWORD PTR [ebx+0xc], eax
0x8048074 <one+18> lea     ecx, [ebx+0x8]
→ 0x8048077 <one+21> mov     edx, 0x0
0x804807c <one+26> mov     al, 0xb
0x804807e <one+28> int     0x80
0x8048080 <two+0> call    0x8048062 <one>
0x8048085 <two+5> das
0x8048086 <two+6> bound  ebp, QWORD PTR [ecx+0x6e]
0x8048089 <two+9> das
0x804808a <two+10> jae     0x80480f4

```

(10) Then we use the instruction "mov edx, 0" because we don't need to pass environment variables.

(11) Then we use the instruction "mov al, 0xb" to set eax = 0x0000000b because the system call number of execve is 11 (0x0000000b).

```

[ Legend: Modified register | Code | Heap | Stack | String ]
registers
$eax : 0xb
$ebx : 0x00048085 → "/bin/sh"
$ecx : 0x0004808d → 0x00048085 → "/bin/sh"
$edx : 0x0
$esp : 0xbffff720 → 0x00000001
$ebp : 0x0
$esi : 0x00048085 → "/bin/sh"
$edi : 0x0
$ebp : 0x0004807e → <one+28> int 0x80
$eflags: [carry parity adjust zero sign trap INTERRUPT direction overflow RESUME virtualx86 identification]
$cs: 0x0073 $ss: 0x007b $ds: 0x007b $es: 0x007b $fs: 0x0000 $gs: 0x0000

stack
0xbffff720 +0x0000: 0x00000001 ← $esp
0xbffff724 +0x0004: 0xbffff859 → "/home/kaiyu/Lab02/labsh_rel"
0xbffff728 +0x0008: 0x00000000
0xbffff72c +0x000c: 0xbffff875 → "TERM=xterm-256color"
0xbffff730 +0x0010: 0xbffff889 → "SHELL=/bin/bash"
0xbffff734 +0x0014: 0xbffff899 → "USER=root"
0xbffff738 +0x0018: 0xbffff8a3 → "LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so(...)"
0xbffff73c +0x001c: 0xbffffe2b → "SUDO_USER=kaiyu"

code:x86:32
0x8048074 <one+18> lea     ecx, [ebx+0x8]
0x8048077 <one+21> mov     edx, 0x0
0x804807c <one+26> mov     al, 0xb
→ 0x804807e <one+28> int     0x80
0x8048080 <two+0> call    0x8048062 <one>
0x8048085 <two+5> das
0x8048086 <two+6> bound  ebp, QWORD PTR [ecx+0x6e]
0x8048089 <two+9> das
0x804808a <two+10> jae     0x80480f4

```

(12) Then we use the instruction "int 0x80" to invoke kernel actions from a userland process to do the system call.