Київський національний університет імені Тараса Шевченка
факультет радіофізики, електроніки та комп'ютерних систем

Лабораторна робота № 4
Варіант 4

Роботу виконав
студент 3 курсу
КІ-СА
Бондаренко Владислав

Київ 2020

1. Підготовка середовища розробки

Для виконання лабораторної роботи вам знадобиться комп'ютер (віртуальний або фізичний) архітектури AMD64/EM64T із встановленим дистрибутивом ОС Linux (будь-яким).
На систему необхідно встановити GCC, GDB, GNU Make та GNU Binutils.
Створіть окремий каталог, який будете використовувати для виконання лабораторної роботи.
Завантажте в нього файл із символами та програму-заготовку:
defs.h
exit.s

```
[root@g00-s00 Lab4]# wget http://tilde.slu.kiev.ua/cs/asm/defs.h
```

```
[root@g00-s00 Lab4]# wget http://tilde.slu.kiev.ua/cs/asm/exit.s_
```

```
[root@g00-s00 Lab4]# ls
defs.h  exit.s
```

Виконайте асемблювання програми-заготовки та зв'язування:
as -o exit.o -c exit.s
ld -static -o exit exit.o
Пересвідчіться у тому, що виконуваний файл працездатний. Програма повинна нічого не робити і не виводити жодних помилок.

```
[root@g00-s00 Lab4]# as -o exit.o -c exit.s
[root@g00-s00 Lab4]# ld -static -o exit exit.o
[root@g00-s00 Lab4]# ls
defs.h  exit  exit.o  exit.s
```

```
[root@g00-s00 Lab4]# ./exit
[root@g00-s00 Lab4]#
```

2. Автоматизація збірки

Створіть Makefile, який за командою make exit та make all виконає збірку, а за командою make clean очистить об'єктні та виконувані файли.
Модифікуйте Makefile так, щоб опції асемблера та лінкера задавалися змінними ASFLAGS та LDFLAGS.
Додайте опцію асемблера для генерації відлагоджувальних символів DWARF.

Використайте шаблонні правила так, щоб можна було збирати декілька асемблерних файлів в окремі виконувані файли. Це знадобиться при виконанні індивідуального завдання.

```
AS_FLAGS=--gdwarf-2 -o
LD_FLAGS=--static
SOURCES=truth.s
OBJECTS=$(SOURCES:.s=.o)
EXECUTABLE=truth

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
    ld $(LD_FLAGS) $(OBJECTS) -o $@

.s.o:
    as $(AS_FLAGS) $@ -c $<

clean:
    rm -f *.o
```

```
[root@g00-s00 Lab4]# make truth
as --gdwarf-2 -o truth.o -c truth.s
ld --static truth.o -o truth
```

3. Навички відлагоджування

Завантажте одержаний виконуваний файл у відлагоджувач за допомогою команди:
gdb ./exit

```
[root@g00-s00 Lab4]# gdb ./truth
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-119.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /root/Lab4/truth...done.
(gdb) _
```

Встановіть точку зупинки на початок програми (мітка _start):

```
(gdb) b _start
Breakpoint 1 at 0x4000b1: file truth.s, line 15.
```

Запустіть програму.

```
(gdb) run
```

Після зупинки виконання програми перегляньте вміст регістрів:

```
(gdb) i r
rax            0x0        0
rbx            0x0        0
rcx            0x0        0
rdx            0x0        0
rsi            0x0        0
rdi            0x0        0
rbp            0x0        0x0
rsp            0x7fffffffe670   0x7fffffffe670
r8             0x0        0
r9             0x0        0
r10            0x0        0
r11            0x0        0
r12            0x0        0
r13            0x0        0
r14            0x0        0
r15            0x0        0
rip            0x4000b1 0x4000b1 <_start>
eflags         0x202      [ IF ]
cs             0x33       51
ss             0x2b       43
ds             0x0        0
es             0x0        0
fs             0x0        0
gs             0x0        0
```

Переходьте до виконання наступної команди:

```
(gdb)                    n
17          movq %rsp, %rcx /* rcx = *argc */
(gdb) n
19          addq $1, %rbx /* rbx = argc + 1 */
(gdb) _
```

Для виходу із режиму покрокового виконання використовуйте команду continue або c
Програма працюватиме до наступної точки зупинки або до повного чи аварійного завершення.

```
(gdb) c
Continuing.
XDG_VTNR=1
```

4. Індивідуальні завдання

Створіть програму, яка виводить вміст змінних оточення власного процесу на стандартний потік виведення.

Прізвище	Ім'я	Варіант
Бондаренко	Владислав	4

Додаткова довідка

Розміщення параметрів командного рядка та змінних оточення на стеку

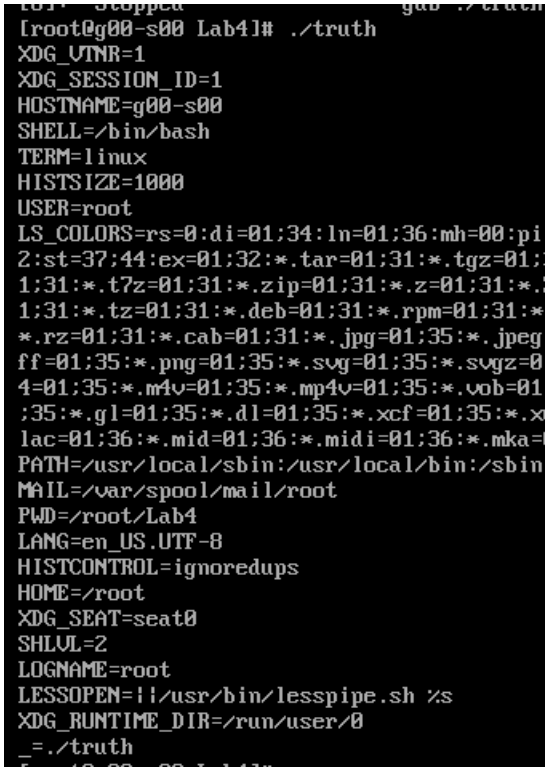
- `0(%rsp)` - `argc`
- `8(%rsp)` - `argv[0]` - name of executable
- ... - argc-1 arguments
- NULL - end of arguments
- ... - `envp[0]` - environment
- ...
- NULL - end of environment

Псевдокод

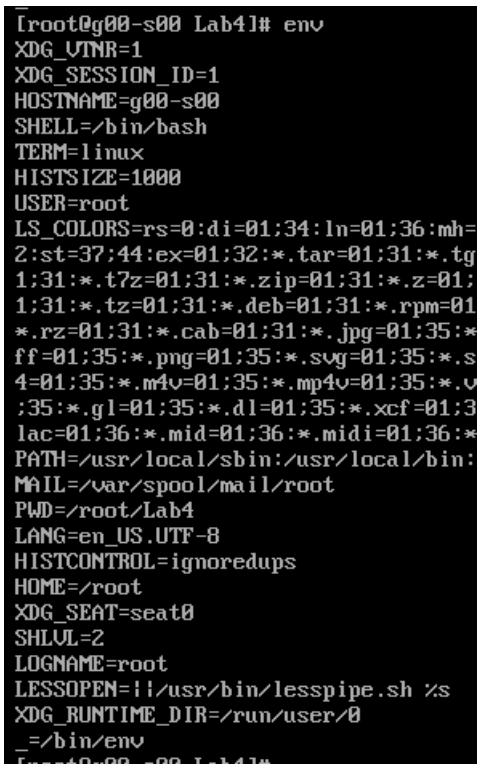
```
int len;
byte *p;
const char *newline = "\n";
main() {
    int argc = *(%rsp)
    char **envp = %rsp + 8 * (argc + 2);
    while(envp != NULL) {
        len = 0;
        p = *envp;
        while(*p != '\0') {
            p++;
            len++;
        }
        write(stdout, len, *envp);
        write(stdout, 1, newline);
    }
}
```

Код програми доступний на github.

Результат виконання програми:



Результат виконання стандартної команди для відображення змінних оточення **env**:



Висновки

В даній лабораторній роботі було ознайомлено з архітектурою x86-64, зокрема з вказівником команд RIP, флаговим регістром RFLAGS та цілочисельними регістрами загального призначення (RSP, RBP, RAX,…)

Також під час роботи було ознайомлено та отримано практичні навички роботи з GNU асемблером, який використовує AT&T синтаксис.

Було виконано автоматизацію збірки за допомогою make-файла, проведено відлагоджування коду з використанням GNU дебагера.