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

Лабораторна робота № 4

Тема: « Кодування асемблером на архітектурі AMD64/EM64T **»**

Роботу виконав
студент 3 курсу
КІ - СА
Гулівець Владислав
Андрійович

Київ 2020

Хід виконання роботи

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.
gcc:

```
[root@g00-s00 Lab4]# gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/libexec/gcc/x86_64-redhat-linux/4.8.5/lto-wrapper
Target: x86_64-redhat-linux
Configured with: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --with
-bugurl=http://bugzilla.redhat.com/bugzilla --enable-bootstrap --enable-shared --enable-threads=posi
x --enable-checking=release --with-system-zlib --enable-__cxa_atexit --disable-libunwind-exceptions
--enable-gnu-unique-object --enable-linker-build-id --with-linker-hash-style=gnu --enable-languages=
c,c++,objc,obj-c++,java,fortran,ada,go,lto --enable-plugin --enable-initfini-array --disable-libgcj
--with-isl=/builddir/build/BUILD/gcc-4.8.5-20150702/obj-x86_64-redhat-linux/isl-install --with-cloog
=/builddir/build/BUILD/gcc-4.8.5-20150702/obj-x86_64-redhat-linux/cloog-install --enable-gnu-indirec
t-function --with-tune=generic --with-arch_32=x86-64 --build=x86_64-redhat-linux
Thread model: posix
gcc version 4.8.5 20150623 (Red Hat 4.8.5-39) (GCC)
```

gdb:

```
[root@g00-s00 Lab4]# gdb -v
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/>.
```

GNU MAKE, GNU BINUTILS:

```
[root@g00-s00 Lab4]# yum install devtoolset-7-toolchain c
```

Я встановив ці розширення за допомоги цієї групи-пакетів.

Створіть окремий каталог, який будете використовувати для виконання лабораторної роботи.
Завантажте в нього файл із символами та програму-заготовку:

- defs.h
- exit.s

Виконайте асемблювання програми-заготовки та зв'язування:

- as -o exit.o -c exit.s
- ld -static -o exit exit.o

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

Як бачимо програма компілюється та зв’язалась. Програма не видає жодних помилок.

Пересвідчіться у тому, що виконуваний файл працездатний. Програма повинна нічого не робити і не виводити жодних помилок.

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.

```
CMP=as
LNR=ld
ASFLAGS=-g -c
ASFLAGSDBG=--gdwarf2
LNRFLAGS=-static
SRCS=$(wildcard *.s)
OBJS=$(SRCS:.s=.o)
PROGS=$(OBJS:.o=)

.PHONY: all clean
.SILENT: clean

exit: exit.s
	$(CMP) $(ASFLAGS) -o $($<:.s=.o) -c $<
	$(LNR) $(LNRFLAGS) -o $@ $($<:.s=.o)

all: $(PROGS) $(OBJS)

$(PROGS): %: %.o
	$(LNR) $(LNRFLAGS) -o $@ $<
$(OBJS): %.o: %.s
	$(CMP) $(ASFLAGS) -o $@ -c $<

clean:
	rm -rf *.o
	ls | grep -v '\.\|Makefile' | xargs rm -rf
```

```
[root@g00-s00 Lab4]# ls
check_around.s  check_around.txt  defs.h  exit.s  Makefile
[root@g00-s00 Lab4]# make all
Makefile:20: warning: overriding recipe for target `exit'
Makefile:14: warning: ignoring old recipe for target `exit'
as -g -c -o exit.o -c exit.s
ld -static -o exit exit.o
as -g -c -o check_around.o -c check_around.s
ld -static -o check_around check_around.o
[root@g00-s00 Lab4]# ls
check_around    check_around.s    defs.h  exit.o  Makefile
check_around.o  check_around.txt  exit    exit.s
[root@g00-s00 Lab4]# _
```

3. Навички відлагоджування

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

```
(gdb) gdb ./exit
```

Встановіть точку зупинки на початок програми (мітка _start):
b _start

```
(gdb) b _start
Breakpoint 1 at 0x400078
```

Запустіть програму
run

```
(gdb) run
Starting program: /Lab4/./exit
```

Після зупинки виконання програми перегляньте вміст регістрів:
info registers або i r

```
(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            0x7fffffffe680   0x7fffffffe680
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            0x400078 0x400078 <_start>
eflags         0x202    [ IF ]
cs             0x33     51
ss             0x2b     43
ds             0x0      0
es             0x0      0
fs             0x0      0
gs             0x0      0
(gdb)
```

Переходьте до виконання наступної команди:
next або n

```
(gdb) n
Single stepping until exit from function _start,
which has no line number information.
[Inferior 1 (process 1319) exited normally]
```

Для виходу із режиму покрокового виконання використовуйте команду continue або c
Програма працюватиме до наступної точки зупинки або до повного чи аварійного завершення.
Для перегляду адресного простору процесу скористайтесь командою x, наприклад:
x/16gx 0x12345678

```
(gdb) x/16gx 0x12345678
```

4. Індивідуальні завдання

Варіант: **4. Правда про своє оточення**

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

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

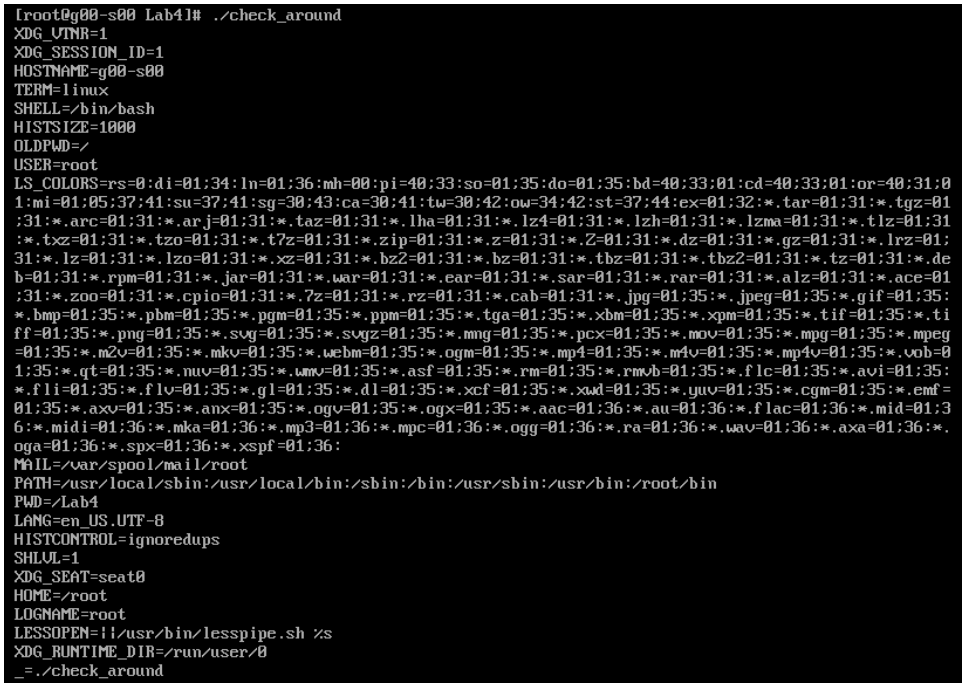
```
int len;
byte *p;
const char *newline = "\n";
main() {
   int argc = *(%rsp)
   char **envp = %rsp + 8 * (argc + 2);
   while(envp != NULL) {
      len = 0;
      p = *envp;
      while(*p != '\0') {
         p++;
         len++;
```

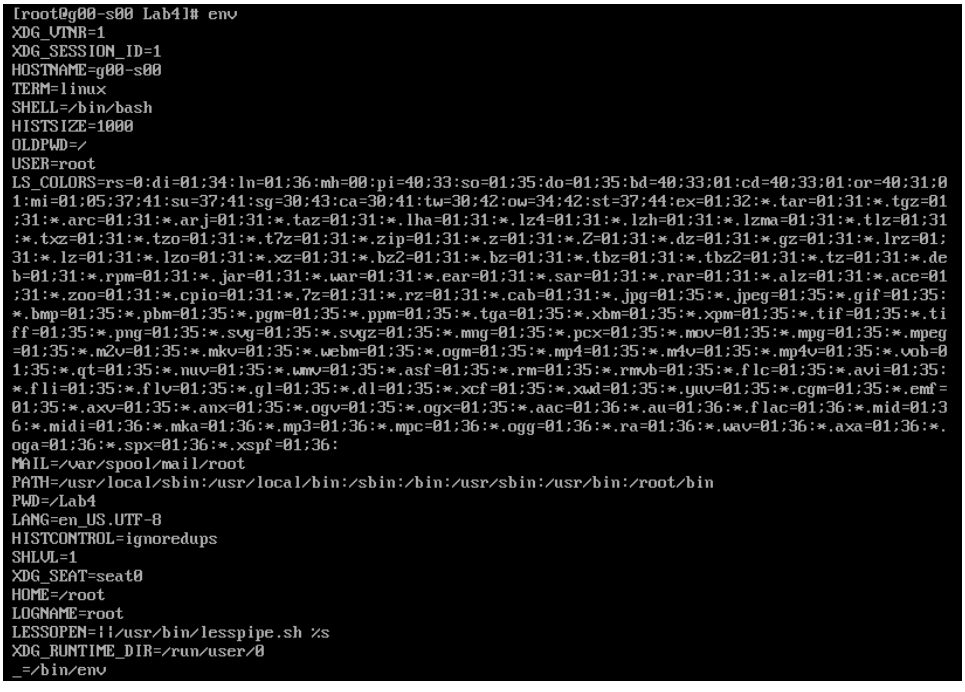
```
        }
        write(stdout, len, *envp);
        write(stdout, 1, newline);
    }
}
```

Код моєї програми

Перевірка

Порівняйте результат виконання із результатом команди env.





Висновок: В цій лабораторній роботі мною було зроблена програма на асемблері, продемонстрована її робота. Також я навчився роботи автоматизації збірки.