

Министерство высшего образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)
Кафедра безопасности информационных систем (БИС)

КОМБИНИРОВАННЫЕ ПРОГРАММЫ. СВЯЗЫВАНИЕ РАЗНОЯЗЫКОВЫХ
МОДУЛЕЙ

Отчет по лабораторной работе №3
по дисциплине «Системное программирование»

Студентка гр.737-1

_____ Агеева В.С.

____.____.2021г

Принял

Руководитель

доцент кафедры БИС

_____ Романов А.С.

____.____.2021г

Томск 2021

1 Введение

Цель работы: познакомиться с основными способами передачи параметров подпрограмм, особенностями передачи управления между модулями, научиться писать комбинированные программы, в которых модули Ассемблера вызываются из модулей, написанных на высокоуровневых языках программирования.

Задание:

1. Изучить краткие теоретические сведения, материалы лекций по теме практического занятия и приведенные выше примеры программ.
2. Все действия, описанное далее выполнять в docker контейнере, образ - любой привычный linux.
3. Решение основной задачи реализовать на ассемблере в виде процедуры (рисунок 1.1). Далее получить объектный модуль. Вызывать из кода на C++ процедуру из объектного файла и скомбинировать их в один исполняемый файл.

1. Напишите программу, в которой создается два числовых массива одинакового размера. Необходимо вычислить сумму попарных произведений элементов этих массивов. Так, если через a_k и b_k , обозначить элементы массивов (индекс $0 \leq k < n$), то необходимо вычислить сумму $\sum_{k=0}^{n-1} a_k b_k$

Рисунок 1.1 – Задание на лабораторную работу

2 Ход работы

Для выполнения лабораторной работы создадим 3 файла через nano:

- с кодом ассемблера;
- с кодом на высокоуровневом языке с использованием вставки ассемблера;
- докер (рисунок 2.1).

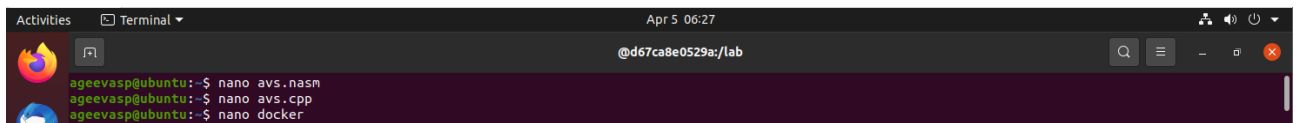


Рисунок 2.1 – Создание файлов

На рисунках 2.2 – 2.3 показан код ассемблера и код на высокоуровневом языке.

```

global foo

section .text
; DWORD foo(DWORD dst, DWORD src, DWORD n)
foo:
    push ebp
    mov ebp, esp

    sub esp, 4 ; объявляем локальную переменную

    mov edi, [ebp+8] ; адрес первого массива
    mov esi, [ebp+12] ; адрес второго массива
    mov ecx, [ebp+16] ; количество элементов
    mov DWORD [ebp-4], 0
.lp:
    mov eax, [esi]
    mul DWORD [edi]
    add DWORD [ebp-4], eax ; сохраняем результат в локальной переменной
    add esi, 4 ; переходим к следующему элементу в первом массиве
    add edi, 4 ; переходим к следующему элементу во втором массиве
    loop .lp

    mov eax, DWORD [ebp-4] ; сохраняем результат из локальной переменной
    add esp, 4 ; удаляем локальную переменную

    mov esp, ebp
    pop ebp
    ret

```

Рисунок 2.2 – Код на языке ассемблера

```
#include <stdio>
#include <stdint>
#include <stddef>

extern "C"
{
int foo(int *arr1, int *arr2, size_t size);
}

int main(void) {

    int arr1[] = {1, 2, 3};
    int arr2[] = {1, 2, 3};

    size_t size = sizeof(arr1) / sizeof(int);

    printf("%d\n", foo(arr1, arr2, size));

    return 0;
}
```

Рисунок 2.3 – Код на языке C

Соберем и запустим наш образ докер файла (рисунки 2.4 – 2.5).

```

ageevasp@ubuntu:~$ docker build -t lab3 .
Sending build context to Docker daemon 88.79MB
Step 1/5 : FROM fedora
--> 26056ca25aff
Step 2/5 : RUN dnf update -y
--> Using cache
--> 31d9ef96e4c8
Step 3/5 : RUN dnf install g++ gcc gdb nasm glibc-devel.i686 libstdc++-devel.i686 -y
--> Running in a4f9ab8f4546
Last metadata expiration check: 0:10:21 ago on Mon Apr 5 10:19:08 2021.
Dependencies resolved.
=====
Package                                Arch      Version              Repository           Size
=====
Installing:
gcc-c++                                x86_64    10.2.1-9.fc33        updates              11 M
gdb                                     x86_64    10.1-4.fc33          updates              125 k
glibc-devel                            i686     2.32-4.fc33          updates              1.0 M
libstdc++-devel                        i686     10.2.1-9.fc33        updates              2.0 M
nasm                                    x86_64    2.15.03-2.fc33       fedora               438 k
Installing dependencies:
binutils                               x86_64    2.35-18.fc33         updates              5.4 M
binutils-gold                          x86_64    2.35-18.fc33         updates              774 k
boost-regex                             x86_64    1.73.0-10.fc33       updates              288 k
cpp                                     x86_64    10.2.1-9.fc33        updates              9.4 M
ctags                                   x86_64    5.8-30.fc33          fedora               176 k
elfutils-debuginfod-client             x86_64    0.183-1.fc33         updates              34 k
gc                                       x86_64    8.0.4-4.fc33         fedora               104 k
gcc                                     x86_64    10.2.1-9.fc33        updates              30 M
gdb-headless                           x86_64    10.1-4.fc33          updates              4.1 M
glibc                                   i686     2.32-4.fc33          updates              3.4 M
glibc-devel                            x86_64    2.32-4.fc33          updates              1.0 M
glibc-headers-x86                      noarch    2.32-4.fc33          updates              482 k
guile                                   x86_64    5:2.0.14-21.fc33     fedora               3.4 M
guile22                                x86_64    2.2.7-1.fc33         updates              6.5 M
isl                                     x86_64    0.16.1-12.fc33       fedora               876 k
kernel-headers                         x86_64    5.11.11-200.fc33     updates              1.2 M
libbabeltrace                          x86_64    1.5.8-3.fc33         fedora               189 k
libgcc                                  i686     10.2.1-9.fc33        updates              108 k
libc_u                                x86_64    67.1-4.fc33          fedora               9.7 M

```

Рисунок 2.4 – Сборка докер файла

```

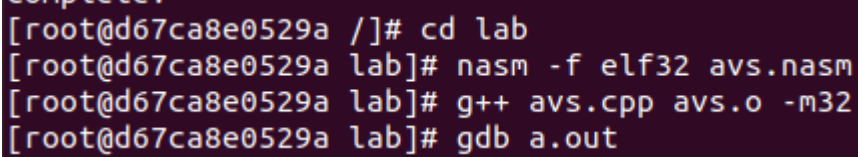
Complete!
Removing intermediate container a4f9ab8f4546
--> 9eb729aa6701
Step 4/5 : COPY avs.cpp /lab/
--> b13ca7fb3f2e
Step 5/5 : COPY avs.nasm /lab/
--> 0b65398a25bd
Successfully built 0b65398a25bd
Successfully tagged lab3:latest
ageevasp@ubuntu:~$ docker run --rm -it lab3
[root@1537477f145c /]# cd lab

```

Рисунок 2.5 – Запуск докер файла

Для того, чтобы код на высокоуровнем языке программирования скомпилировался, необходимо ввести следующее: `dnf install glibc-devel.i686 libstdc++-devel.i686`.

На рисунке 2.6 показано какие нужно выполнять действия дальше.

A terminal window with a dark background and light-colored text. The text shows a series of commands and their prompts in a root shell. The commands are: changing directory to 'lab', assembling 'avs.nasm' to 'avs.o' using nasm, compiling 'avs.cpp' to 'a.out' using g++, and launching the debugger 'gdb' on 'a.out'.

```
[root@d67ca8e0529a /]# cd lab  
[root@d67ca8e0529a lab]# nasm -f elf32 avs.nasm  
[root@d67ca8e0529a lab]# g++ avs.cpp avs.o -m32  
[root@d67ca8e0529a lab]# gdb a.out
```

Рисунок 2.6 – Компиляция кодов и запуск gdb

3 Заключение

В ходе лабораторной работы были приобретены знания и навыки написания комбинированных программ, в которых модуль, написанный на ассемблере, вызывается программой, написанной на языке высокого уровня.

Для выполнения задания по варианту была написана и протестирована программа на ассемблере, затем адаптирована и интегрирована в виде ассемблерной вставки в программу, написанную на языке Си.

Исходные коды программ и Dockerfile приложены в архиве к отчету, а также загружены на GitHub ().