Министерство науки и высшего образования Российской Федерации
Федерального государственное бюджетное образовательное учреждение
высшего образования
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра комплексной информационной безопасности электронно-
вычислительных систем (КИБЭВС)

КОМБИНИРОВАННЫЕ ПРОГРАММЫ. СВЯЗЫВАНИЕ РАЗНОЯЗЫКОВЫХ МОДУЛЕЙ

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

Выполнил
Студент гр. 738-1
_____ Вдовина И.Е.
_____.__.2022

Принял
М.н.с. ИСИБ
_____ Калинин Е. О.
_____.__.2022

Томск 2022

Введение

Целью работы является 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.

3 Вариант: Debian.

2 ХОД РАБОТЫ

Для начала создадим файл под названием «lab3» и напишем код для 3 варианта (рисунок 2.1).

```
#include <iostream>
#include <ctime>

using namespace std;

int main()
{

    srand(time(0));
    int n=6;
    long long array[n][n];
    for (int i = 0; i < n ;i++)
    {
        for (int j = 0; j < n; j++)
        {
            array[i][j]=rand()%9+1;
            cout<<array[i][j]<< " ";
        }
         cout << endl;
    }

   int sum=0;

   for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            sum+=array[i][j]*array[i][j];
        }

    }

    cout << "Сумма полученная в c++: " << sum << endl;

    int a=0;

    long long *ptr_array=&array[0][0];
    long long *end_array=&array[n-1][n-1];

        asm (
        "movq $0, %%rax\n\t"
        "movq %[ptr_array], %%rbx \n\t"
        "subq $8, %%rbx \n\t"
        "movq $0, %%rcx \n\t"

        "start1: \n\t"
                "addq $8, %%rbx \n\t"
                "movq (%%rbx), %%rax\n\t"
                "mulq %%rax \n\t"
                "addq %%rax, %%rcx \n\t"
                "cmpq %%rbx, %[end_array] \n\t"
                "jne start1 \n\t"
        "movq %%rcx, %%rax \n\t"
                :"=a"(a)
                :[ptr_array]"m"(ptr_array), [end_array]"m"(end_array)
                :"%rbx"
        );
    cout << "Сумма полученная в ассемблере " << a << endl;
    return 0;
}
```

Рисунок 2.1 – Код программы

Сделаем файл исполняемым с помощью команды «g++ name -static -o name» и проверим работоспособность программы (рисунок 2.2).

```
root@vie738:/home/vdovina# g++ lab3.cpp -static -o lab3
root@vie738:/home/vdovina# ./lab3
7 6 2 3 3 3
8 6 8 3 4 3
2 9 7 9 1 1
9 4 3 6 6 2
5 8 9 9 1 7
7 6 1 6 8 3
Сумма полученная в с++: 1209
Сумма полученная в ассемблере 1209
```

Рисунок 2.2 – Проверка работоспособности программы

Следующем шагом является создание «Dockerfile» и собрать образ (рисунок 2.3 – 2.4).

```
1 FROM debian
2 COPY lab3.cpp .
3 RUN apt-get update
4 RUN apt-get install -y gcc
5 RUN apt-get install -y g++
6 RUN g++ lab3.cpp -static -o lab3
7 CMD ./lab3
```

Рисунок 2.3 – Содержание Dockerfile

```
root@vie738:/home/vdovina# docker build -t lab3 .
Sending build context to Docker daemon  193.7MB
Step 1/7 : FROM debian
 ---> d40157244907
Step 2/7 : COPY lab3.cpp .
 ---> 0cf4e087336a
Step 3/7 : RUN apt-get update
 ---> Running in 5ff8a44f3ff7
Get:1 http://security.debian.org/debian-security bullseye-security InRelease [44.1 kB]
Get:2 http://deb.debian.org/debian bullseye InRelease [116 kB]
Get:3 http://deb.debian.org/debian bullseye-updates InRelease [39.4 kB]
Get:4 http://security.debian.org/debian-security bullseye-security/main amd64 Packages [124 kB]
Get:5 http://deb.debian.org/debian bullseye/main amd64 Packages [8182 kB]
Get:6 http://deb.debian.org/debian bullseye-updates/main amd64 Packages [2596 B]
Fetched 8508 kB in 5s (1578 kB/s)
Reading package lists...
Removing intermediate container 5ff8a44f3ff7
 ---> c1aff7a4c550
Step 4/7 : RUN apt-get install -y gcc
 ---> Running in 9f9095bacceb
Reading package lists...
Building dependency tree...
Reading state information...
The following additional packages will be installed:
  binutils binutils-common binutils-x86-64-linux-gnu cpp cpp-10
  fontconfig-config fonts-dejavu-core gcc-10 libasan6 libatomic1 libbinutils
  libbrotli1 libbsd0 libc-dev-bin libc-devtools libc6 libc6-dev libcc1-0
  libcrypt-dev libctf-nobfd0 libctf0 libdeflate0 libexpat1 libfontconfig1
  libfreetype6 libgcc-10-dev libgd3 libgomp1 libisl23 libitm1 libjbig0
  libjpeg62-turbo liblsan0 libmd0 libmpc3 libmpfr6 libnsl-dev libpng16-16
  libquadmath0 libtiff5 libtirpc-dev libtsan0 libubsan1 libwebp6 libx11-6
```

Рисунок 2.4 – Собранный образ Dockerfile

Далее проверим работоспособность программы, запустим образ с помощью команды «docker run -it name» (рисунок 2.5).

```
root@vie738:/home/vdovina# docker run -it lab3
9 6 1 4 1 4
1 6 3 5 1 5
5 4 9 2 4 1
1 9 7 3 6 9
6 3 4 4 1 5
6 9 1 4 1 2
Сумма полученная в c++: 890
Сумма полученная в ассемблере 890
```

Рисунок 2.5 – Запуск образа

Заключение

В ходе выполнения данной работы было произведено ознакомление с основными способами передачи параметров подпрограмм, особенностями передачи управления между модулями; изучение комбинированных программ, в которых модули Ассемблера вызываются из модулей, написанных на высокоуровневых языках программирования.

Был создан docker-контейнер, с предустановленными компонентами компиляции и редактирования кода, необходимыми для написания программ на языке C++. В данной среде были написана и протестирована необходимая по заданию программа.

Приложение А

Репозиторий на GitHub

https://github.com/GreenRakes/SP