Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Кафедра комплексной информационной безопасности электронно-вычислительных систем (КИБЭВС)

М.В. Сорокин

Методическое пособие по дисциплине

«Системное программирование»

Тема № 2 – Комбинированные программы. Связывание модулей разных языков программирования. Ассемблерные вставки.

**Цель работы:**

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

**Теоретические сведения:**

Процедуры, написанные на языке Assembler оснащены возможностью получения данных из вызывающей процедуры, написанной на высокоуровневом языке программирования, также они имеют возможность возвращения и не возвращения результата своей работы. Приведем аналогию на примере языка программирования С++.

Процедура (подпрограмма) — это основная функциональная единица декомпозиции (разделения на несколько частей) некоторой задачи. Процедура представляет собой группу команд для решения конкретной подзадачи и обладает средствами получения управления из точки вызова задачи более высокого приоритета и возврата управления в эту точку. В простейшем случае программа может состоять из одной процедуры. Процедуру можно определить и как правильным образом оформленную совокупность команд, которая, будучи однократно описана, при необходимости может быть вызвана в любом месте программы.

Функция – процедура, способная возвращать некоторое значение. Процедуры ценны тем, что могут быть активизированы в любом месте программы. Процедурам могут быть переданы некоторые аргументы, что позволяет, имея одну копию кода в памяти и изменять ее для каждого конкретного случая использования, подставляя требуемые значения аргументов.

Для описания последовательности команд в виде процедуры в языке ассемблера используются две директивы: PROC и ENDP.

Синтаксис описания процедуры:

ИмяПроцедуры PROC язык расстояние

; тело процедуры

ИмяПроцедуры ENDP

Рассматривая пример, можно заметить, что в строчке 17, имеются два входных параметра: *FirstNumericalValue* и *SecondNumericalValue,* которые в данном случае образуют выходной параметр, в строке 18, являющийся результатом суммирования. Язык Ассемблер не обладает таким функционалом для выполнения аналогичных действий, но у него есть другие методы передачи параметров.

Листинг кода на С++:

#include "iostream"

using namespace std;

int sum (int FirstNumericalValue, int SecondNumericalValue);

main(){

int FirstNumericalValue, SecondNumericalValue, result;

cout << "Введите первое число:" << endl;

cin >> FirstNumericalValue;

cout << "Введите второе число:" << endl;

cin >> SecondNumericalValue;

cout << "Результат:"<< endl << sum(FirstNumericalValue,SecondNumericalValue) << endl;

return 0;

}

int sum (int FirstNumericalValue, int SecondNumericalValue){

return FirstNumericalValue + SecondNumericalValue;

}

Рассмотрим несколько имеющихся способов передачи параметров в процедуры.

1. **Передача параметров через регистры.** Если данных передается немного, то самый быстрый и простой способ – передать параметры через регистры. Если же данных много, или они представляют сложные структуры типа массива или записи, то использовать регистры не рационально. В качестве примера, можно привести использование передачи параметров через регистры, при вызове функции прерывания BIOS.
2. **Передача данных путем обращения к памяти.** При таком способе обмена данными вызываемая и вызывающая процедуры обращаются напрямую к данным по их условным именам. Способ оформления такого обращения зависит от того, как организована программа: находятся ли основная программа и процедура в одном исходном модуле (файле) или же в разных модулях (файлах). Существует две вариации исхода событий, а то есть: совместная трансляция процедуры и основной программы или же раздельная трансляция процедуры и основной программы.

***Рассмотрим первый случай (совместная трансляция).***

При совместной трансляции вся программа помещается в один файл, т.е. представляет собой один исходный модуль, который транслируется за один вызов транслятора. В этом случае формируется единое адресное пространство программы, и все имена данных программы видимы в процедуре. Схема совместной трансляции представлено на рисунке 1.

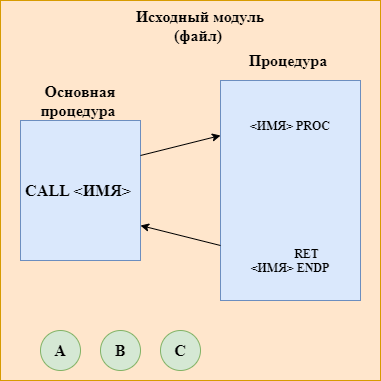


Рисунок 1 – Схема совместной трансляции

Далее рассмотрим код (пример программы) для более подробного понимания работы совместной трансляции (использование среды разработки RadASM, траслятор MASM32).

Пример 1 – Определение суммы двух чисел.

Листинг на Ассемблер:

.386 *; модель процессора*

.model **flat**, stdcall *;используемая модель памяти*

option casemap: none *;в названиях переменных не будет учитываться регистр*

include c:/masm32/include/windows.**inc** *;подключение необходимых библиотек*

include c:/masm32/include/user32.**inc**

include c:/masm32/include/masm32.**inc**

include c:/masm32/include/kernel32.**inc**

includelib c:/masm32/lib/user32.lib

includelib c:/masm32/lib/kernel32.lib

includelib c:/masm32/lib/masm32.lib

**.DATA** *;Определяет начало инициализированного сегмента данных;*

szCaption **db** 'Сумма двух чисел', 0 *;Наименование MessageBox’a*

format **db** " Результат вычислений = %d",0 *; Строка форматирования*

A **DWORD** 55 *;Переменная А типа "DWORD" выделение памяти 4 байта;*

B **DWORD** 34 *;Переменная B типа "DWORD" выделение памяти 4 байта;*

**.DATA**? *;Определяет сегмент данных, в котором располагаются неинициализированные данные;*

D **DWORD** ? *;Неизвестное значение переменной D типа "DWORD"*

buff **db** 100 DUP **(**?**)** *;Переменная типа “byte” с размеров в 100 и имеющая значение NULL*

.**CODE** *;Определяет сегмент программного кода и заканчивает предыдущий сегмент, если таковой имеется;*

**Start**:

**lea** **EDX**, D *;Помещение указателя (ссылки) на переменную D в регистре EDX*

**mov** **EAX**, A *;Помещение в регистр EAX значения переменной A*

**mov** **EBX**, B *;Помещение в регистр EBX значения переменной B*

**call** SumDword *;Вызов процедуры SumDword*

Invoke wsprintfA, addr buff, addr format, **EAX** *;Перевод числа в строковый тип данных*

invoke MessageBox, NULL,addr buff,offset szCaption, MB\_OK *;Вывод в диалоговое окно полученного значения*

Invoke ExitProcess,0 *;Корректное завершение процесса*

SumDword PROC *;Описание процедуры "SumDword";*

**add** **EAX**, **EBX** *;Прибавление к значению регистра EAX значения регистра EBX*

**mov** **[EDX]**, **EAX** *;Запись значения EAX в регистр EDX*

**ret** *;Выполняет возврат из ближней процедуры;*

SumDword ENDP *;Определяет конец процедуры и имеет имя, аналогичное имени в директиве*

END **Start** *; Этой директивой завершается любая программа на ассемблере. В роли необязательного операнда здесь выступает метка (или выражение), определяющая адрес, с которого начинается выполнение программы (точка входа в программу).*

***Рассмотрим второй случай (раздельная трансляция).***

При раздельной трансляции процедуры описываются в разных файлах, транслируются отдельно и объединяются в единую программу на этапе компоновки. Каждый файл в этом случае – отдельный модуль со своим адресным пространством. Поэтому необходимо указать компоновщику внутренние имена модуля, к которым будет происходить обращение из других модулей, и внешние имена, которые определены в других модулях, но к которым есть обращение из данного модуля. Для этого предусмотрены специальные директивы. Схема работы раздельной трансляции представлена на изображении 2.

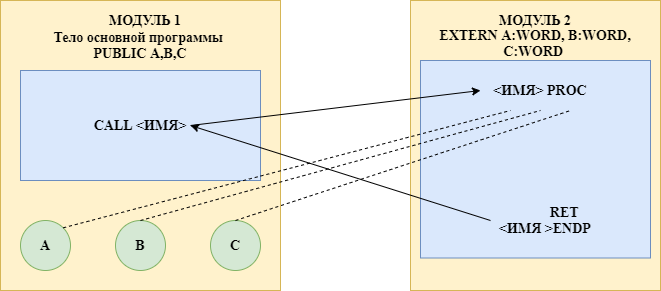


Рисунок 2 – Схема раздельной трансляции

Директива PUBLIC описывает внутренние имена, к которым возможно обращение извне:

***PUBLIC [<Язык>] <Имя> [, <Язык>] <Имя>…***

где *<Язык> -* параметр, определяющий конвенцию о связи, т.е. особенности формирования внутренних имен глобальных переменных и процедур, а *<Имя> -* символическое имя, которое должно быть доступно в других модулях.

***EXTERN [Язык]<Имя>[(<Псевдоним>)] :<Тип>***

***[, [<Язык>] <Имя>[(<Псевдоним>)] :<Тип> …***

где *<Имя>* - символическое имя, используемое в процедуре, но не описанное в ней, *<Тип>* - определяется для различных типов имен следующим способом:

идентификаторы (типы данных): ***BYTE***, ***WORD*,** ***DWORD***;

имя процедуры, метка, указатели: ***NEAR***, ***FAR***;

константа, определенная посредством ‘=’ или ‘***EQU***’: ***ABS***.

При этом, если в одной процедуре имя описано как ***EXTERN***, то в другой оно должно быть описано как ***PUBLIC***.

Универсальная директива ***EXTERNDEF*** описывает любое имя, которое описано в одном модуле, а используется в других. В зависимости от обстоятельств может интерпретироваться как ***PUBLIC*** или ***EXTERN***:

Пример 2 – Определение суммы двух чисел (использование среды разработки RadASM, траслятор MASM32). Раздельное исполнение основного тела программы и вызываемой процедуры (отдельного модуля).

Листинг на Ассемблер:

.386 *;модель процессора*

.model **flat**, stdcall *;используемая модель памяти*

option casemap: none *;в названиях переменных не будет учитываться регистр*

include c:/masm32/include/windows.**inc** *;подключение необходимых библиотек*

include c:/masm32/include/user32.**inc**

include c:/masm32/include/masm32.**inc**

include c:/masm32/include/kernel32.**inc**

includelib c:/masm32/lib/user32.lib

includelib c:/masm32/lib/kernel32.lib

includelib c:/masm32/lib/masm32.lib

**.DATA** *;Определяет начало инициализированного сегмента данных;*

szCaption **db** 'Сумма двух чисел', 0 *; Наименование MessageBox’a*

format **db** " Результат вычислений = %d",0 *; Строка форматирования*

A **DWORD** 53 *;Переменная А типа "DWORD" выделение памяти 4 байта;*

B **DWORD** 34 *;Переменная B типа "DWORD" выделение памяти 4 байта;*

**.DATA**? *;Определяет сегмент данных, в котором располагаются неинициализированные данные;*

D **DWORD** ? *;Неизвестное значение переменной D типа "DWORD"*

buff **db** 100 DUP **(**?**)** *;Переменная типа “byte” с размеров в 100 и имеющая значение NULL*

**PUBLIC** A, B, C *; Обьявление внутренних имен;*

**EXTERN** SumDword:near *;Обьяление внешних имен;*

.**CODE** *; Определяет сегмент программного кода и заканчивает предыдущий сегмент, если таковой имеется;*

**Start**: **call** SumDword

Invoke wsprintfA, addr buff, addr format, **EAX** *;Перевод числа в строковый тип данных*

invoke MessageBox, NULL,addr buff,offset szCaption, MB\_OK *;Вывод в диалоговое окно полученного значения*

Invoke ExitProcess,0 *;Корректное завершение процесса*

End **Start**.

**Отдельный модуль программы:**

.386

.model **flat**, stdcall

option casemap: none

.**CODE**

**EXTERN** A: **DWORD**, B: **DWORD**, D: **DWORD** *; Идентификатор переменных*

SumDword PROC c *;Описание процедуры "SumDword"*

**mov** **EAX**, A *; Запись в регистр EAX числа А (1 - е число);*

**add** **EAX**, B *; Сложение Числа A с числом B;*

**mov** D, **EAX** *; Результат сложения, число D, запись в регистр EAX;*

**pop** **EAX** *; Забрать значение из стека*

**RET** *; Выполняет возврат из ближней процедуры;*

SumDword ENDP *; Определяет конец процедуры и имеет имя, аналогичное имени в директиве PROC;*

END *; Этой директивой завершается любая программа на ассемблере. В роли необязательного операнда здесь выступает метка (или выражение), определяющая адрес, с которого начинается выполнение программы (точка входа в программу)*

***Примечание*** – Создание многомодульных программ с использованием RADAsm выполняют следующим образом:

1) добавление модуля осуществляется с использованием пункта меню **Проект/Добавить новый/Модуль;**

2) ассемблирование модуля осуществляется при активизации пункта меню **Создать/Assemble Modules**. Для активизации этого пункта меню необходимо в меню **Проект/Настройка проекта**, пометить галочкой в списке меню таблицы выделить **Assemble Modules**

3) после получения объектного модуля до компоновки необходимо этот модуль также добавить к проекту, используя **Проект/Добавить существующие/Объектные модули**

1. **Передача параметров через таблицу адресов.** В этом случае в памяти вызывающей программы создается специальная таблица адресов параметров. В таблицу перед вызовом процедуры записывают адреса передаваемых данных. Затем адрес самой таблицы заносится в один из регистров и управление передается вызываемой процедуре. Вызываемая процедура сохраняет в стеке содержимое всех регистров, которые собирается использовать, после чего выбирает адреса переданных данных из таблицы, выполняет требуемые действия и заносит результат по адресу, переданному в той же таблице. На рисунке 3 показана таблица адресов параметров.

****

Рисунок 3 – Таблица адресов параметров

Следующим шагом рассмотрим пример решения задачи – суммирования элементов массива.

Пример 3 – Суммирование элементов массива. Массив и его размер определяются в основной программе, суммирование элементов выполняет процедура . Результат сложения возвращается в основную программу. Раздельное исполнение основного тела программы и вызываемой процедуры (отдельного модуля). Использование среды разработки RadASM, траслятор MASM32.

Листинг на ассемблере:

.386 *;модель процессора*

.model **flat**, stdcall *;используемая модель памяти*

option casemap: none *;в названиях переменных не будет учитываться регистр*

include c:/masm32/include/windows.**inc** *;подключение необходимых библиотек*

include c:/masm32/include/user32.**inc**

include c:/masm32/include/masm32.**inc**

include c:/masm32/include/kernel32.**inc**

includelib c:/masm32/lib/user32.lib

includelib c:/masm32/lib/kernel32.lib

includelib c:/masm32/lib/masm32.lib

**.data** *;Определяет начало инициализированного сегмента данных;*

szCaption **db** 'Сумма двух чисел', 0 *;Наименование MessageBox’a*

format **db** " Результат вычислений = %d",0 *; Строка форматирования*

ary SWORD 1,2,3,4,5,6,7,8,9 *;Входной массив данных;*

count **DWORD** 9 *;Размера входного массива данных;*

**.data**? *;Определяет сегмент данных, в котором располагаются неинициализированные данные;*

sum SWORD ? *;Неизвестное значение sum типа "SWORD";*

tabl **DWORD** 3 DUP **(**?**)** *;Таблица адресов параметров;*

**EXTERN** masculc:near *;Обьяление внешних имен;*

.**CODE**

**Start**:

*;Формирование таблицы адресов параметров;*

**mov** tabl, offset ary

**mov** tabl+4, offset count

**mov** tabl+8, offset

**mov** **EBX**, offset tabl

**call** masculc

**XOR** **EAX**,**EAX**

Invoke wsprintfA, addr buff, addr format, **AX** *;Перевод числа в строковый тип данных*

invoke MessageBox, NULL,addr buff,offset szCaption, MB\_OK *;Вывод в диалоговое окно полученного значения*

invoke ExitProcess,0 *;Корректное завершение процесса*

End **Start**.

**Отдельный модуль программы:**

.386

.MODEL **flat**,

stdcall OPTION

CASEMAP:NONE

.**CODE**

masculc PROC c *;Указана конвенция языка C;*

**push** **AX** *;Сохранение регистров;*

**push** **ECX**

**push** **EDI**

**push** **ESI**

*;Использование таблицы адресов параметров;*

**mov** **ESI**, **[EBX]** *;Адрес массива;*

**mov** **EDI**, **[**EVX+4**]** *;Адрес размера;*

**mov** **ECX**, **[EDI]** *;Размер массива;*

**mov** **EDI**, **[EBX**+8**]** *;Адрес результата;*

*;Суммирование элементов массива*

**xor** **AX**, **AX** *;Самоуничтожение результата в регистре AX с помощью логического отрицания;*

cycl: **add** **AX**, **[ESI]** *;Добавление к регистру AX значения из регистра ESI;*

**add** **ESI**, 2 *;Добавление к регистру ESI значение 2;*

**loop** cycl *;Цикл*

*;Формирование результатов вычислений*

**mov** **[EDI]**, **AX**

**pop** **ESI** *;Восстановление регистров;*

**pop** **EDI**

**pop** **ECX**

**pop** **AX**

**pop** **EBP**

**ret** 12

mascucl ENDP

END

1. **Передача параметров в потоке кода.** В этом необычном методе передаваемые процедуре данные размещаются прямо в коде программы, сразу после команды *CALL*. Чтобы прочитать параметр, процедура должна использовать его адрес, который автоматически передается в стеке как адрес возврата из процедуры. Разумеется, функция должна будет изменить адрес возврата на первый байт после конца переданных параметров перед выполнением команды *RET*. Передача параметров в потоке кода, так же, как и передача параметров в стеке в обратном порядке (справа налево), позволяет передавать различное число параметров. Но этот метод - единственный, позволяющий передать по значению параметр различной длины. Доступ к параметрам, переданным в потоке кода, несколько медленнее, чем к параметрам, переданным в регистрах, глобальных переменных или стеке.
2. **Передача параметров через стек.** Наиболее распространенным способом передачи данных в практике программирования процессоров рассматриваемого типа является передача параметров в стеке. Именно этот способ принят в качестве базового и его используют языки высокого уровня. Параметры помещают в стек командой *PUSH*, после чего управление передается вызываемой процедуре. Доступ к параметрам, хранящимся в стеке, из вызываемой процедуры осуществляют через регистр *EBP*. В этот регистр помещают адрес вершины стека, копируя его из регистра указателя стека *ESP*, а затем этот регистр используют как базовый при адресации параметров. Для обеспечения корректного возврата в вызывающую процедуру старое значение регистра *EBP* помещают в стек первой командой процедуры. Параметры в стеке, адрес возврата и старое значение *EBP* вместе называют *фреймом активации процедуры*. Вызываемая процедура, зная структуру стека, извлекает параметры в соответствующие регистры, выполняет над ними операции и записывает результат, используя адрес, переданный в стеке.

***Примечание*** – Следует помнить, что в стек можно поместить 2 или 4 байта. Если в стек надо поместить параметр размером 1 байт, то помещают 2 байта, но байт со старшим адресом не используется. Если задействуется функция с различным числом параметров (переменным), то данный процесс отразится только на размере области параметров, потому что каждый параметр будет помещен в стек, а удаление параметров будет выполнять вызывающая программа. Так же стоит заметить, что возвращаемые значения должны быть записаны в следующие регистры:

char, short, int, enum, near – в регистр AX;

указатели far, huge и прочие 4 – х байтовые величины – в регистры DX:AX;

float, double – в регистры TOS и ST (0) сопроцессора;

struct – записывается в память, а в регистр записывается указатель (структуры длинной в 1 и 2 байта возвращаются в AX, а 4 байта – в DX:AX).

Параметры помещаются в стек сразу перед вызовом процедуры. Именно этот метод используют языки высокого уровня, такие как *С* и *Pascal*. Для чтения параметров из стека обычно используют не команду *POP*, а регистр *ВР*, в который помещают адрес вершины стека после входа в процедуру:

При передаче параметров через стек возникает два вопроса: – в каком порядке записывать параметры в стек; – кто (вызывающая или вызываемая процедура) должен удалять параметры из стека. В обоих вариантах есть свои плюсы и минусы. Например, если стек освобождает вызываемая процедура по команде *RET*, то код программы получается более коротким. Если за освобождение стека отвечает вызывающая программа, то становится возможным вызов нескольких процедур с одними и теми же значениями параметров просто последовательными командами *CALL*. Первый способ более строгий, он используется в языке *Pascal*. Второй, дающий больше возможностей для оптимизации, – в языках *Cи* и *C++*. Вопрос о порядке записи параметров в стек для ассемблера не столь важен, так как и записывают и извлекают параметры подпрограммы на ассемблере. Главное – чтобы этот вопрос был между ними согласован. А вот при взаимодействии ассемблера с языками высокого уровня, следует знать особенности передачи параметров этих языков.

На изображении 4 представлена схема работы передачи параметров через стек.

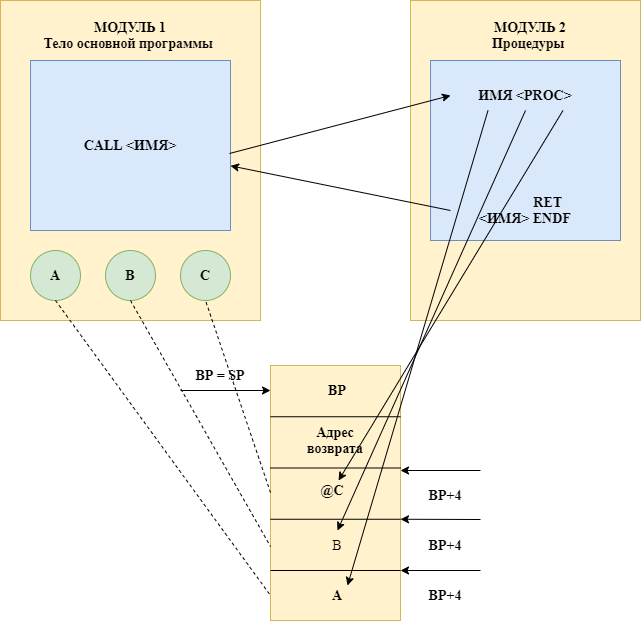


Рисунок 4 – Схема передачи параметров через стек

Также имеется одна особенность внутреннего представления программ на языке *С++*: компилятор языка изменяет используемые имена. Этот процесс, называемый обработкой имен, выполнят сохранение информации о типах аргументов функции, путем модификации имени таким образом, чтобы оно указывало на типы аргументов.

Стоит заметить, что при разработке программ на *С++*, обработка имени происходит автоматически и скрыта от глаз программиста, но если какой – то модуль программы написан на Ассемблере, то программист обязан самостоятельно выполнить обработку имен функций на данном разрабатываемом модуле. Таким образом, чтобы выполнить данные шаги, необходимо знать определённые соглашения, принятые в языке *C++.*

1. В обязательном порядке, перед глобальными именами ставится символ подчеркивания;
2. К именам функций в начало добавляется символ «@», а в конце дописывается знаки «$q» и символы, кодирующие типы параметров функции в виде:

*@ <имя функции> $q <коды типов параметров>*

Соответственно, коды используемых типов параметров, указаны в таблице под номером 1.

Таблица 1 – Коды типов параметров

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Тип** | **Эквивалент** | **Тип** | **Эквивалент** | **Тип** | **Эквивалент** |
| Void | v | Float | F | Long | L |
| Char | zc | Double | D | \*,[] | P |
| Int | i | Short | s | … | E |

Также можно рассмотреть пример использования типов параметров:

fa(char \*b, int \*t[], short d, long z) => @fa$qzcppisl

1. Если есть необходимость убрать (отменить) чувствительность к регистру на языке *C++*, то следует указать опцию “Case sensitive off”.

Обработку имен ассемблерных функций можно и не выполнять, например, чтобы избежать несовместимости с последующими версиями компиляторов, в которых возможны изменения алгоритма этой обработки. С этой целью *С++* позволяет использовать стандартные имена функций Си в программах, написанных на *С++*. Все функции, объявление которых заключено в фигурные скобки, будут иметь имена, соответствующие соглашениям, принятым в языке *Си*. Таким образом, при объявлении в ассемблерном модуле функций, включенных в блок extern “C”, нет необходимости выполнять обработку их имен.

Для того чтобы скомпоновать модули на ассемблере с программой, написанной на *Си*, необходимо следовать определенным соглашениям.

При компиляции исходной программы на *Си* создаются следующие сегменты:

− сегмент кода;

− сегмент данных;

− сегмент неинициализированных данных.

Используемая модель памяти влияет не только на тип вызываемой функции и указателей на данные, но и на то, какие сегменты будет использоваться программой.

В таблице 2 приведены имена сегментов, используемые в языке программирования Си для различных моделей памяти.

Таблица 2 – Имена сегментов, используемые различными моделями памяти

|  |  |  |  |
| --- | --- | --- | --- |
| **Модель памяти** | **Сегмент кодов** | **Сегмент инициированных данных** | **Группа сегментов данных, адресуемых DS** |
| Tiny | \_TEXT | \_DATA | DGROUP |
| Small | \_TEXT | \_DATA | DGROUP |
| Compact | \_TEXT | \_DATA | DGROUP |
| Middle | <имя файла>\_TEXT | \_DATA | DGROUP |
| Large | <имя файла>\_TEXT | \_DATA | DGROUP |
| Hugo | <имя файла>\_TEXT | <имя файла>\_DATA | <имя файла>\_DATA |

Си позволяет ассемблеру увеличивать список глобальных переменных, доступных для всех модулей. Это достигается за счет размещения переменных в сегменте данных, отведенном для глобальных переменных, и описания его внутренним public. Имя такой переменной по правилам Си должно начинаться со знака подчеркивания. Прочие модули, использующие данное имя, должны включать его описание как extern (на Ассемблере) или extern (на Си)[1][2].

**Установка среды виртуализации «Oracle Virtual Box»**

Перед началом выполнения работы, необходимо произвести ряд обязательных действий по установке и настройке нужного программного обеспечения. В первую очередь, следует выбрать средство виртуализации (Oracle Virtual Box [7], VMware Workstation [8]). Следующим шагом после выбора средств виртуализации, необходимо выполнить установку.

Рассмотрим этапы установки на примере Oracle Virtual Box.

Первым шагом, необходимо открыть Oracle Virtual Box, в главном меню выбрать «Создать», как показано на рисунке 5.

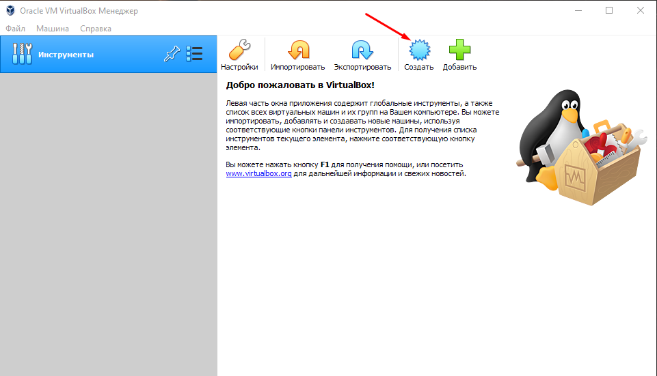


Рисунок 5 – создание виртуальной машины

Затем, следующим шагом нужно выбрать предпочитаемую операционную систему (OC) и нажать «Далее» (3). Для этого, необходимио выбрать папку, в которой будут находиться файлы виртуальной машны (1), затем ввести название виртуальной машины (2). В данном случае, это дистрибутив Ubuntu [9], как показано на изображении 6.

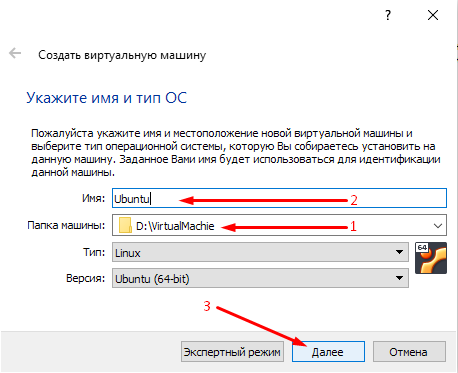


Рисунок 6 – Выбор операционной системы

После выбора OC, программа предложит предварительно выбрать параметры аппаратной части, а именно объём выделяемой оперативной памяти, как представлено на изображении 7. После выбора необходимого количества, необходимо нажать кнопку «Далее», для дальнейшей настройки виртуальной машины.

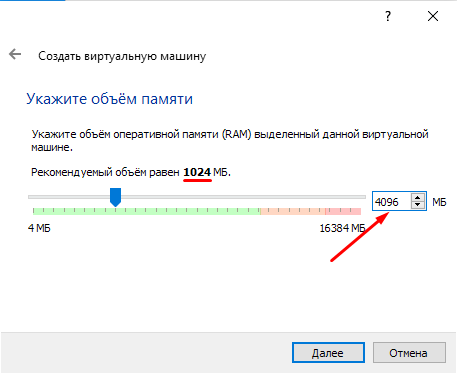


Рисунок 7 – Выбор количества выделяемой оперативной памяти на виртуальную машину

Затем, программа предложит сделать выбор рекомендуемого объёма нового виртуального жесткого диска. В рассматриваемом случае, так как заранее не был создан виртуальный жесткий диск, необходимо выбрать позицию «Создать новый виртуальный жёсткий диск» и также нажать кнопку «Создать», как показано на рисунке 8.

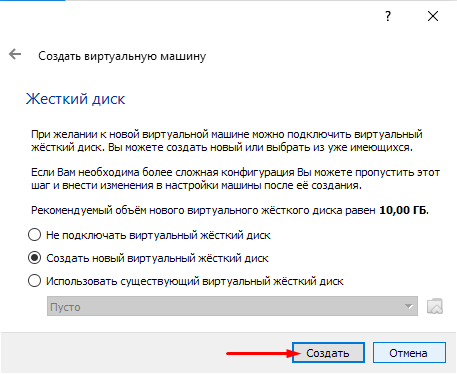


Рисунок 8 – Создание виртуального жёсткого диска

В следующем окне, необходимо выбрать VHD (Virtual Hard Disk. стрелка с 1 указателем) и нажать кнопку «Далее» (2), как представлено на изображении 9.

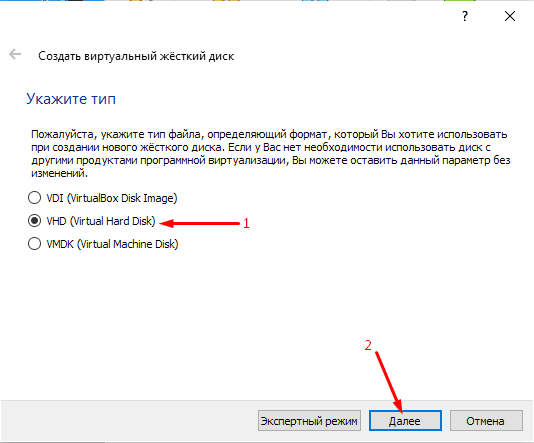


Рисунок 9 – Выбор формата виртуального жёсткого диска

После выбора формата виртуального жёсткого диска, следует выбрать формат хранения (динамический виртуальный жёсткий диск или фиксированный виртуальный жёсткий диск), а затем снова нажать кнопку «Далее», что изображено на рисунке 10.

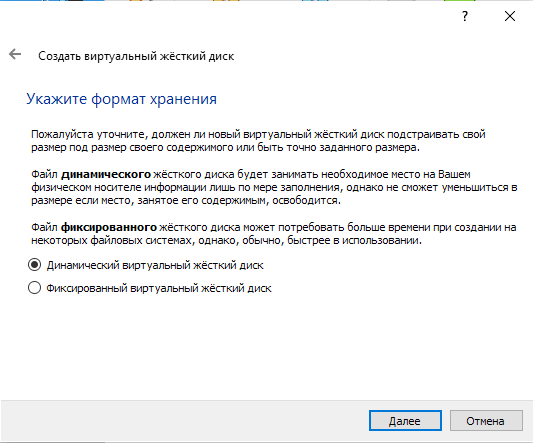


Рисунок 10 – Выбор формата хранения

Следующим шагом, после выбора формата хранения, требуется укаазать размер виртуального жёсткого диска в мегабайтах (1) и вновь нажать кнопку «Создать» (2), как представлено на рисунке 11.

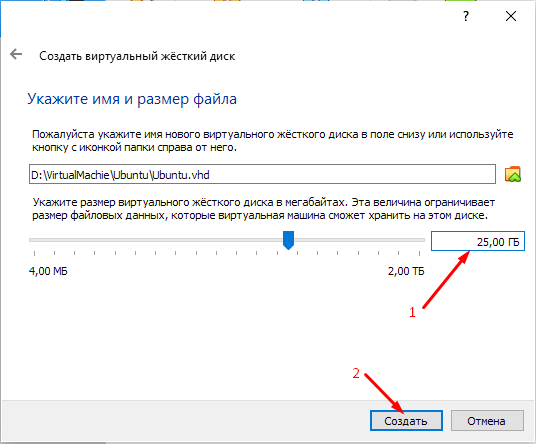


Рисунок 11 – Имя и размер виртуального жёсткого диска

Выполнив ряд предыдущих шагов, в главном меню программы Oracle Virtual Box появится созданная виртуальная машина. После чего, по желанию, можно выполнить более тонкую настройку OC, нажав клавишу «Настроить» на панели быстрого доступа, как представлено на изображении 12.

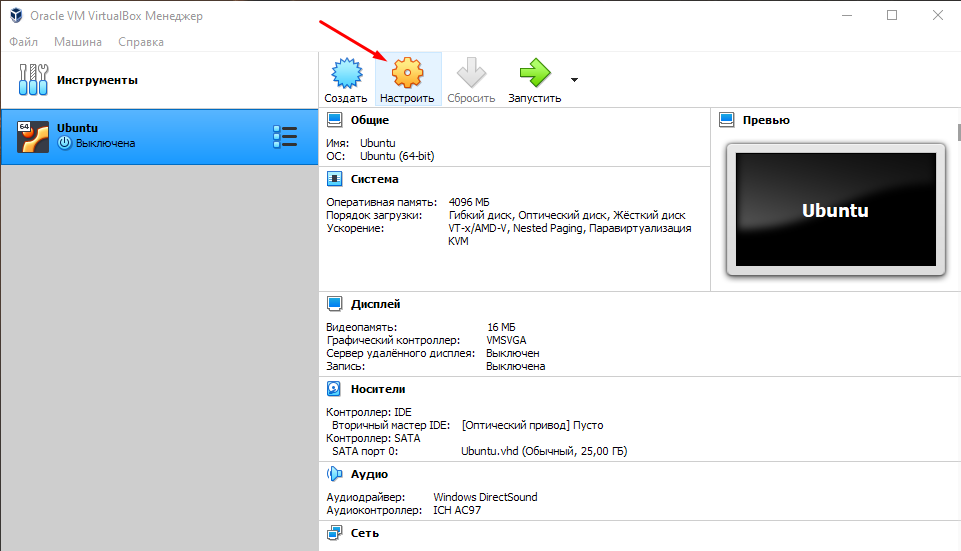


Рисунок 12 – Главное меню программы Oracle Virtual Box

Затем, попав в настройки установленной OC, по необходимости можно произвести более детальную настройку системы. Например, настроить количество процесоров, перейдя во вкладку «Система» (1), затем перейти в дочернюю вкладку «Процессор» (2) и выбрать количество процессоров (3), как представлено на изображениях 13.

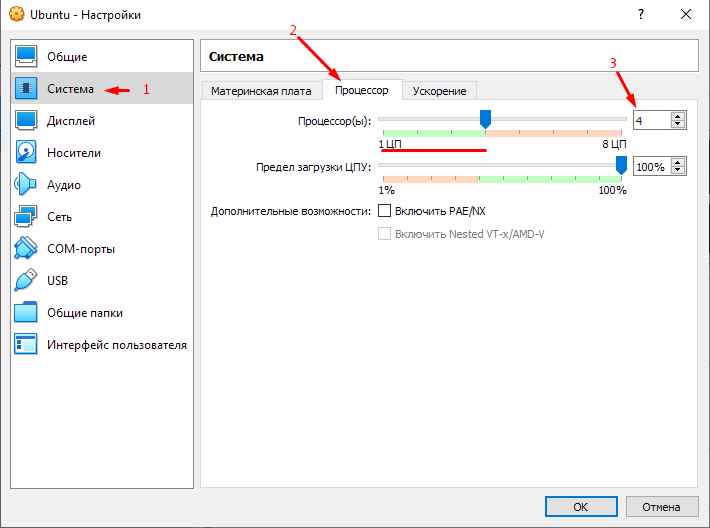


Рисунок 13 – Тонкая настройка OC

Также, по желанию, можно увеличить количество выделяемой видопамяти. Для этого, необходимо перейти во вкладку «Дисплей» (1), а затем в дополнрительную вкладку «Экран» и увеличить количество выделяемой видеопамяти с помощью изменения положения ползунка (2), как представлено на рисунке 14.

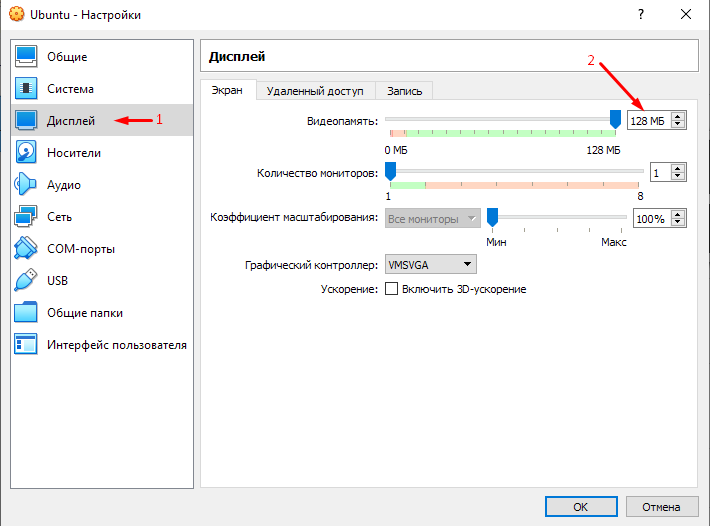


Рисунок 14 – Тонкая настройка OC

После успешной предварительной настройки OC, необходимо выбрать заранее скаченный образ (дистрибутив). Для этого, требуется перейти во вкладку «Носители» (1) и добавить в список приводов оптических дисков (2) скачанный образ, в данном случае Ubuntu 18.04, подтвердив выбор нажатием кнопки «Выбрать» (3), как показано на рисунках 15-17.

После подтверждения выбора, собранная операционная система будет готова к первому запуску.

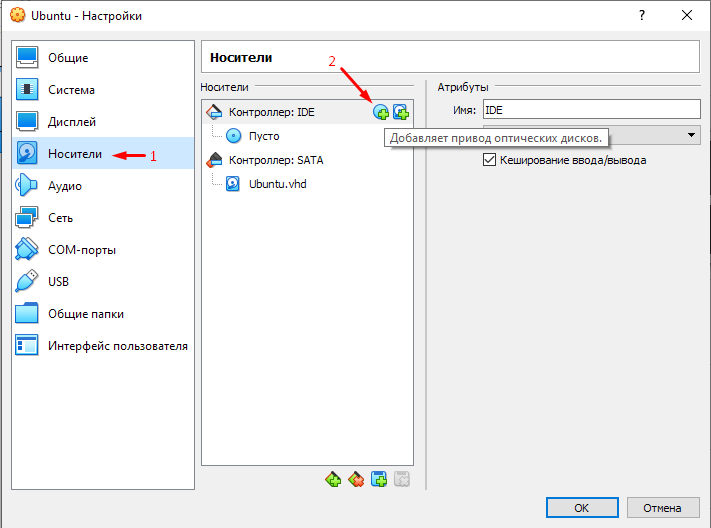


Рисунок 15 – Выбор носителя

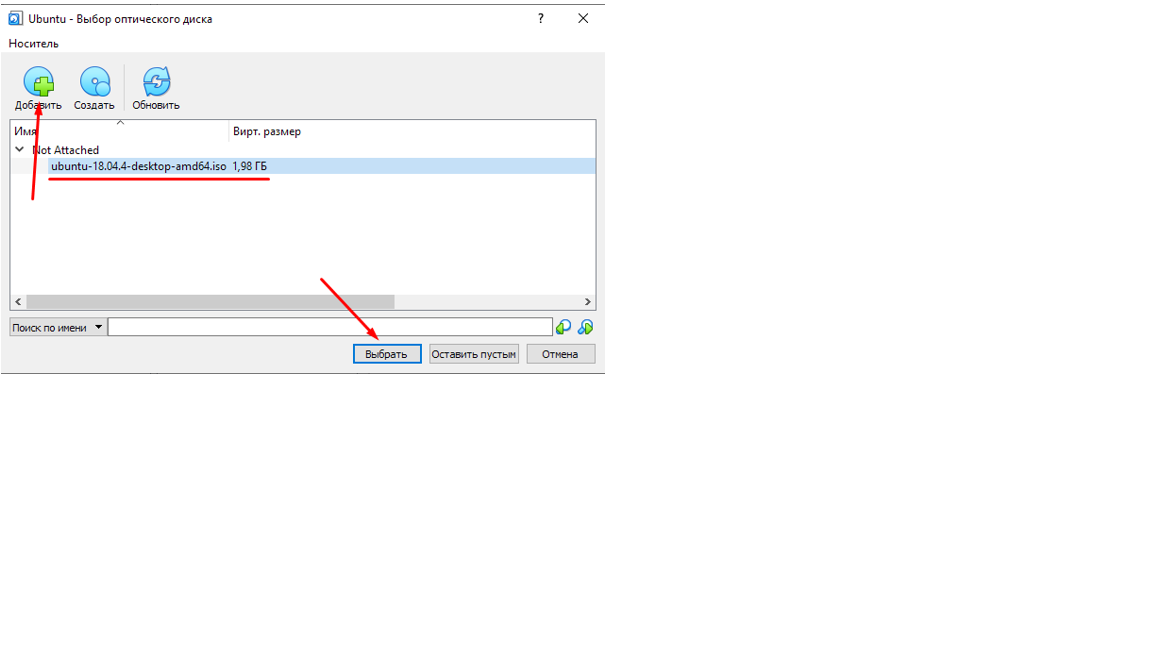


Рисунок 16 – Выбор образа OC (дистрибутива)

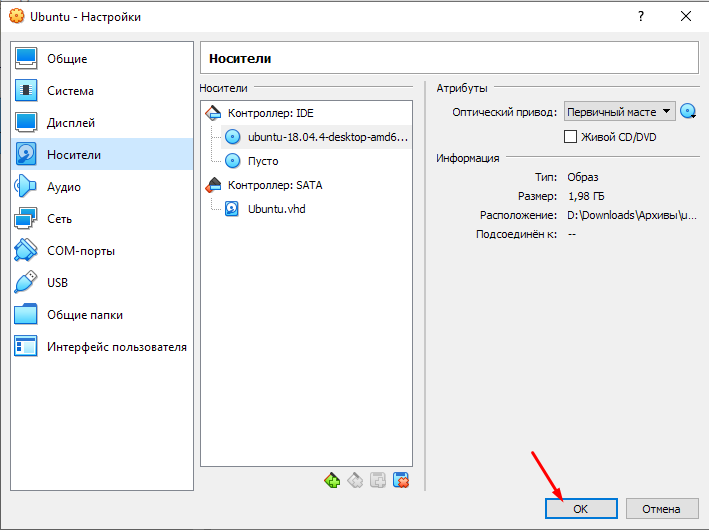


Рисунок 17 – Завершение тонкой настройки OC

**Установка OC Ubuntu**

Следующим этапом, необходимо запустить подготовленный образ, подождать некоторое количество времени, пока не появится экран загрузки операционной системы. После этого, на мониторе появится экран установки Ubuntu, как показано на рисунке 18, в котором будет предложен либо запуск, либо установка OC. Необходимо выбрать Ваш язык, на котором Вам удобнее взаимодействовать с операционной системой (1), а затем нажать «Установка Unumtu» (2) и дождаться перехода в следующее диалоговое окно.

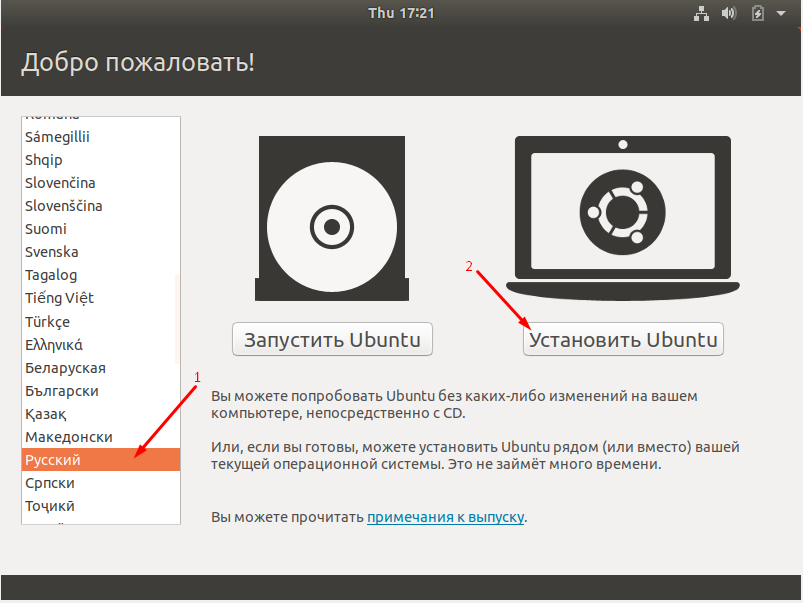


Рисунок 18 – Запуск установки Ubuntu

После нажатия кнопки «Установить Ubuntu», на экране появится диалоговое с выбором раскладки клавиатуры. По умолчанию, в операционной системе установлена английская раскладка клавиатуры, но так можно добавить дополнительную, удобную для Вас раскладку клавиатуры (рис. 19).

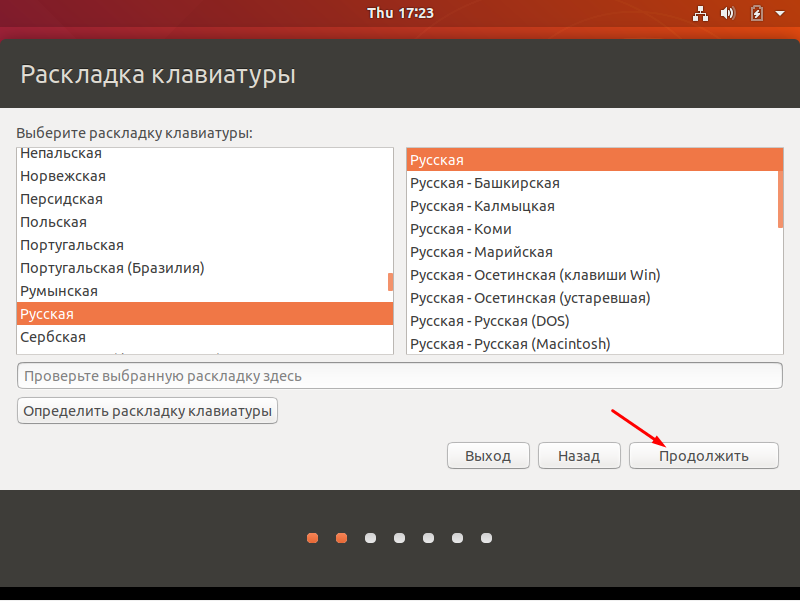


Рисунок 19 – Добавление раскладки клавиатуры

Следующим шагом, необходимо будет выбрать параметры установки операционной системы и дополнительного программного обеспечения. Дабы избежать конфликтных ситуация после установки OC, с аппаратной частью, рекомендуется не выбирать параметр установки (install third – party software...). Следует выбрать «Minimal Installation» (1), если вы не нуждаетесь в дополнительном программном обеспечениии и хотите сохранить больше свободного места на выделенномв виртуальном жёстком диске. Если же для Вас не критично, то можете выполнить полную установку. После выбора, следует нажать кнопку «Продолжить» (2), как показано на изображении 20 и перейти к следующему шагу.

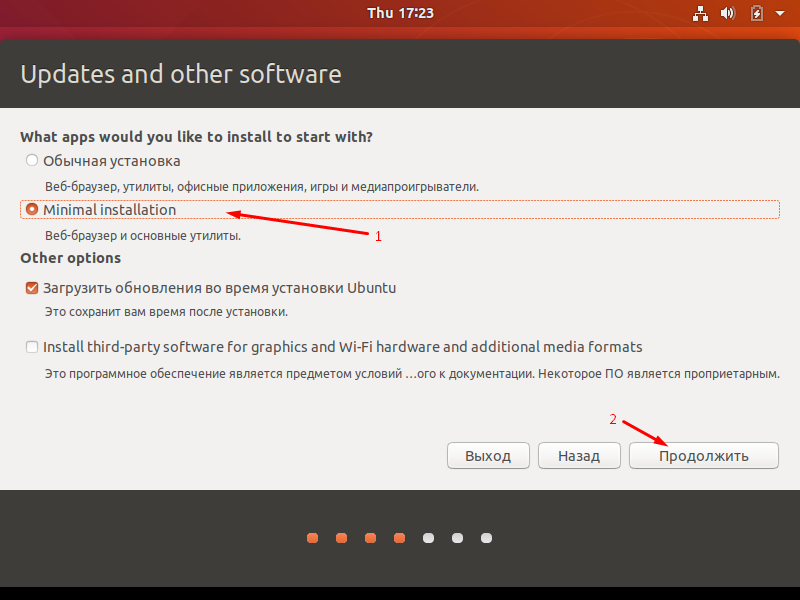


Рисунок 20 – Выбор параметров установки

Далее следует выбрать тип установки OC, как показано на рисунке и нажать кнопку «Установить сейчас», а затем продолжить, как показано на рисунках 21,22.

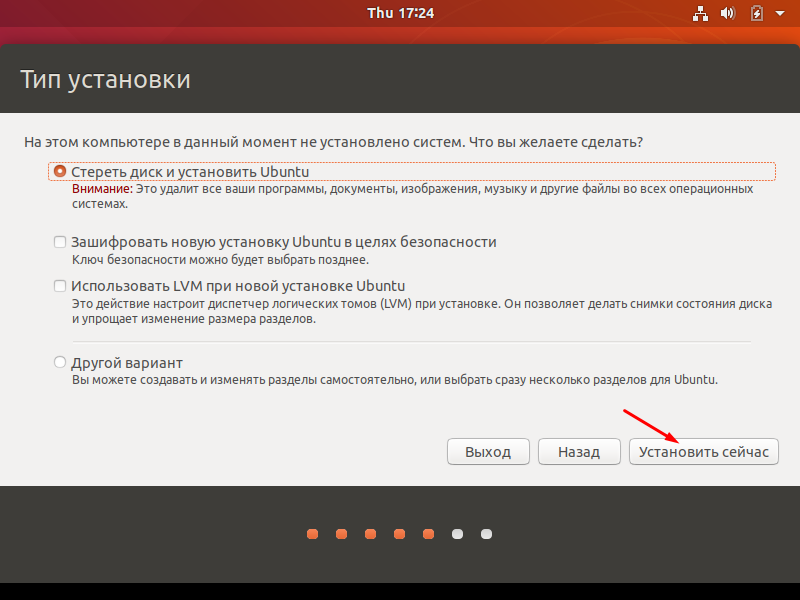


Рисунок 21 – Выбор типа установки установки

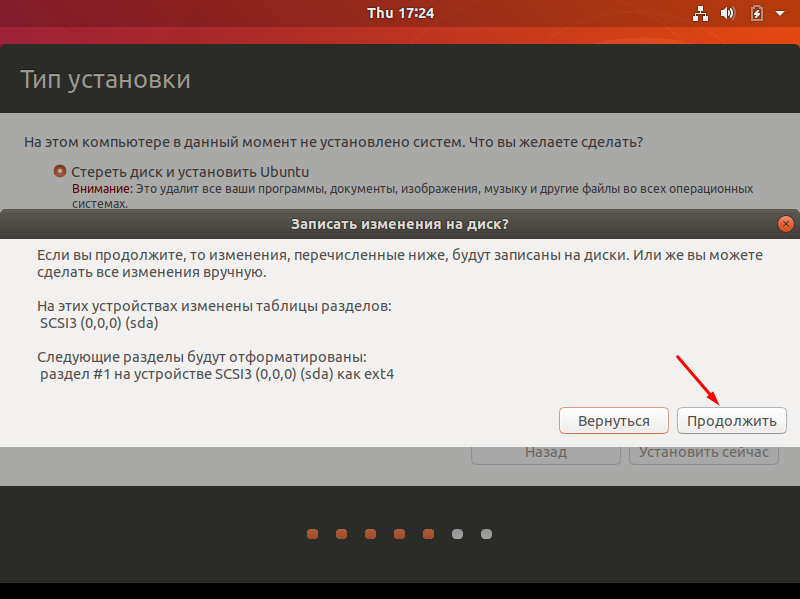


Рисунок 22 – Подтверждение выбранного действия

Далее, в появившемся диалоговом окне, необходимо выбрать предпочитаемый часовой пояс и нажать кнопку «Продолжить» (рис. 23).

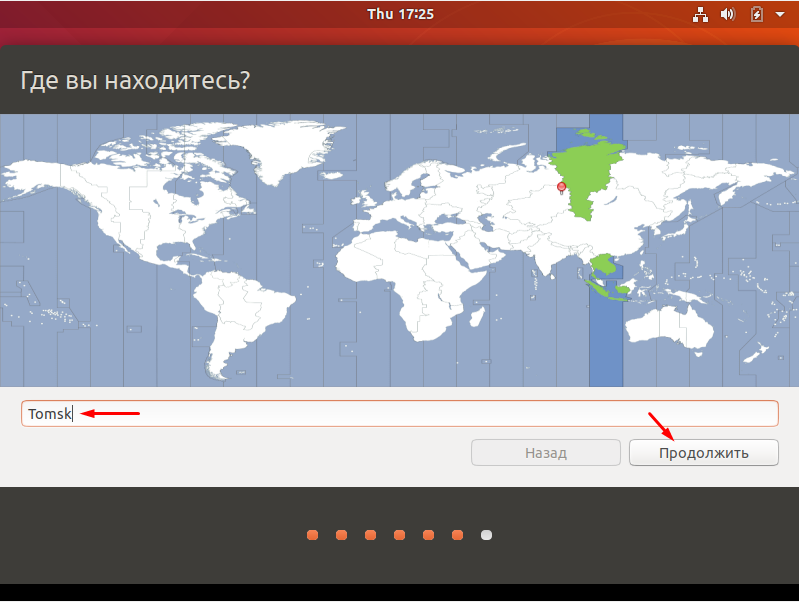


Рисунок 23 – Выбор часового пояса

Следующим шагом, после выбора часового пояса, в открывшемся диалоговом окне, необходимо ввести данные, с помощью которых будет создана учетная запись пользователя в системе (ваше имя (1), логин (2), пароль (3,4)), то есть создать «Учётную записи», а также задать имя персонального компьютера. После подтверждения на правильность введённых данных, требуется нажать кнопку «Продолжить» (5) и перейти к следующему этапу установки (рис. 24).

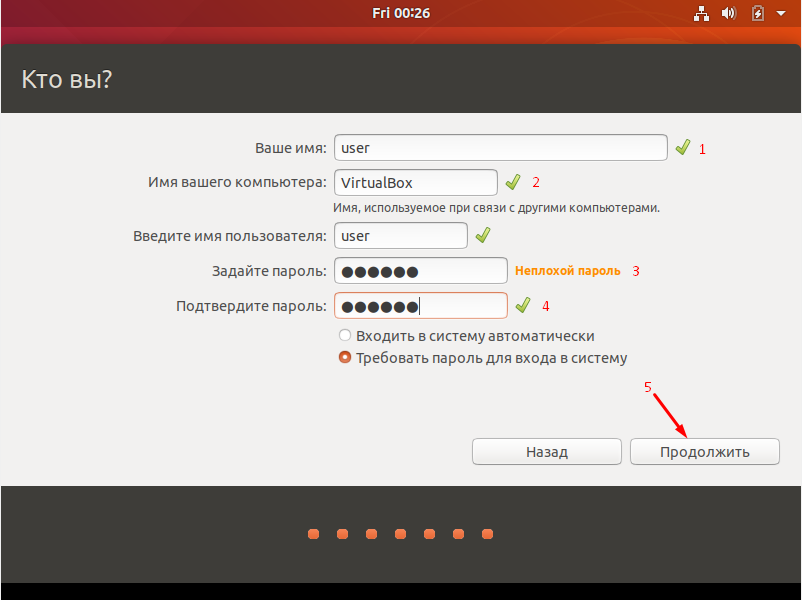


Рисунок 24 – Создание учётной записи

Далее запустится установка OC. После успешной установки операционной системы, на мониторе появится сообщение об успешной установки OC, как показано на рисунке 25.

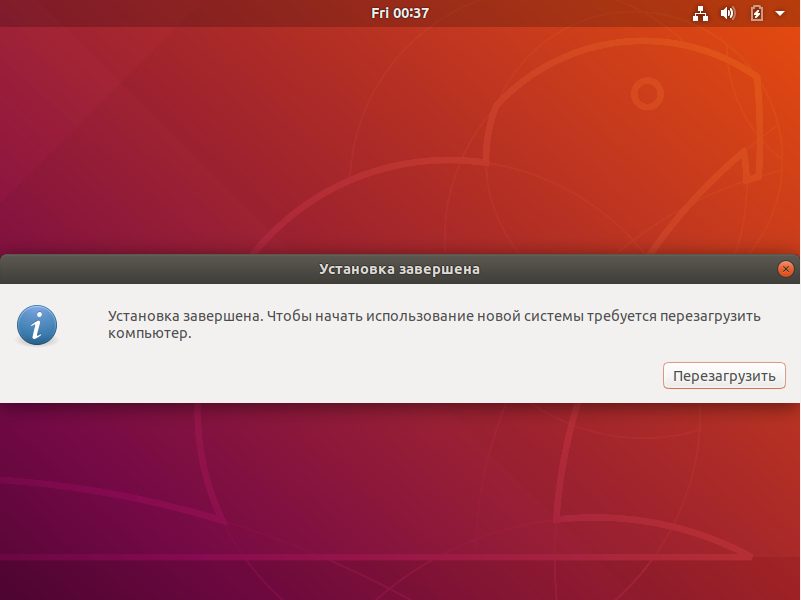


Рисунок 25 – Успешная установка OC

**Установка Docker’a в Unubtu**

Для дальнейшей работы, необходимо установить дополнительное программное обеспечение по заданию – Docker.

Чтобы установить Docker в Unix – подобную систему, существует множество инструкци. Воспользуемся уже отработанным алгоритмом установки данного ПО на предустановленную заранее операционную систему, в нашем случае Ubuntu.

Перед тем как установить Docker Ubuntu необходимо установить дополнительные пакеты ядра, которые позволяют использовать Aufs для контейнеров Docker. С помощью этой файловой системы мы сможем следить за изменениями и делать мгновенные снимки контейнеров. В процессе выполенния команд могут возникать незначиельные ошибки, не стоит предавать им значение:

sudo apt install linux-image-extra-$(uname -r) linux-image-extra-virtual

Также надо установить пакеты, необходимые для работы apt по https. При выполнение данной команды необходимо подтверждение с помощью клавиши «Enter»:

sudo apt install apt-transport-https ca-certificates curl software-properties-common

После того как вся подготовка завершена и вы убедились что ваша система полностью готова, можно перейти к установке. Далее будем устанавливать программу из официального репозитория разработчиков. Для этого сначала необходимо добавить ключ репозитория:

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –

Затем добавьте репозиторий docker в систему:

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"

sudo apt update && apt-cache policy docker-ce

А после совершить установку Docker в Ubuntu:

sudo apt install -y docker-ce

Теперь установка Docker и всех необходимых компонентов завершена. Первой командой добавили ключ репозитория разработчиков, второй - сам репозиторий. Далее обновили списки пакетов и сообщили системе что можно заменять системные пакеты на пакеты из этого репозитория.

Чтобы завершить установку осталось добавить пользователя в группу docker. Иначе при запуске утилиты будет получена ошибка подключения к сокету.

Для начала, наеобходимо создать группу пользователей Docker в Ubuntu следующей командой:

sudo groupadd docker

Далее для добавления необходимо выполнить следующую команду:

sudo usermod -aG docker $(whoami)

Следующим шагом проверим работоспособность сервиса Docker с помощью команды:

sudo systemctl status docker

Также, работа в Docker, не обходится без утилиты управления контейнерами Docker Compose, следовательно её тоже необходимо с помощью последовательного выполенния следующих команд:

sudo curl -L "https://github.com/docker/compose/releases/download/1.25.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose

Утилита была загружена из официального сайта и теперь есть возможность посмотреть её версию с помощью команды[3]:

docker-compose --version

**Чтобы пользоваться Docker’ом, в обязательном порядке, необходимо выполнить перезагрузку операционной системы, на которую производилась установка!!!**

**Утилита Docker. Основные команды**

Все действия с контейнерами выполняются утилитой Docker. Ее можно запускать от имени вашего пользователя после того, как он был добавлен в группу программы. Синтаксис утилиты очень прост:

**docker опции команда опции\_команды аргументы**

**Часть основных команд Docker’a, представлена на изображении 26.**

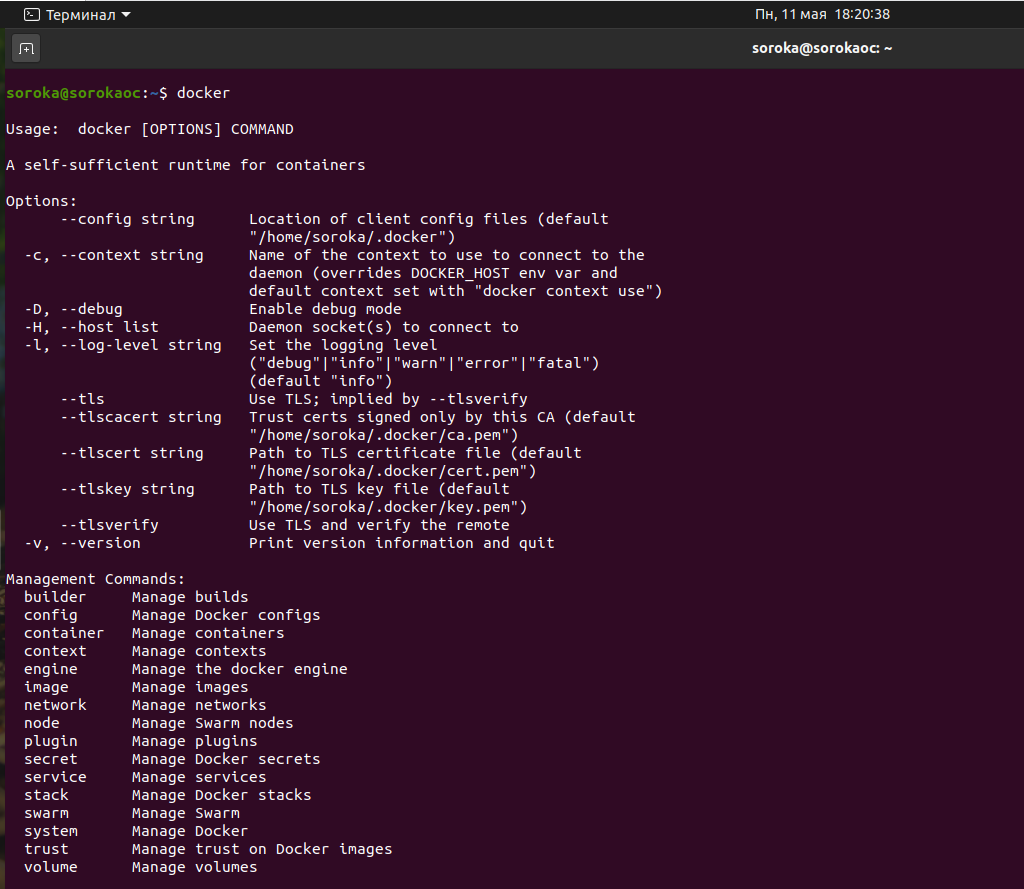


Рисунок 26 – Основные команды Docker’a

1. **-D** - включить режим отладки;

2. **-H**- подключиться к серверу, запущенному на другом компьютере;

3. **-l** - изменить уровень ведения логов, доступно: debug,info,warn,error,fatal;

4. **-v**- показать версию;

5. **--help**- вывести справку по команде или утилите в целом.

Далее рассмотрим все команды, которые можно использовать в программах:

* **attach** - подключиться к запущенному контейнеру;
* **build** - собрать образ из инструкций dockerfile;
* **commit** - создать новый образ из изменений контейнера;
* **cp** - копировать файлы между контейнером и файловой системой;
* **create** - создать новый контейнер;
* **diff** - проверить файловую систему контейнера;
* **events** - посмотреть события от контейнера;
* **exec** - выполнить команду в контейнере;
* **export** - извлечь содержимое контейнера в архив;
* **history** - посмотреть историю изменений образа;
* **images** - список установленных образов;
* **import** - создать контейнер из архива tar;
* **info** - посмотреть информацию о системе;
* **inspect** - посмотреть информацию о контейнере;
* **kill** - остановить запущенный контейнер;
* **load** - загрузить образ из архива;
* **login** - авторизация в официальном репозитории Docker;
* **logout** - выйти из репозитория Docker;
* **logs** - посмотреть логи контейнера;
* **pause** - приостановить все процессы контейнера;
* **port** – проброс портов для контейнера;
* **ps** - список запущенных контейнеров;
* **pull** - скачать образ контейнера из репозитория;
* **push** - отправить образ в репозиторий;
* **restart** - перезапустить контейнер;
* **rm** - удалить контейнер;
* **run** - выполнить команду в контейнере;
* **save** - сохранить образ в архив tar;
* **search** - поиск образов в репозитории по заданному шаблону;
* **start** - запустить контейнер;
* **stats** - статистика использования ресурсов контейнером;
* **stop** - остановить контейнер;
* **top** - посмотреть запущенные процессы в контейнере;
* **unpause** - проложить выполнение процессов в контейнере.

Также стоит уделить внимание флагам которые работают с командой «run», рассмотрим их:

* **-e** - переменные окружения для команды;
* **-h** - имя хоста контейнера;
* **-i** - интерактивный режим, связывающий stdin терминала с командой;
* **-m** - ограничение памяти для команды;
* **-u** - пользователь, от имени которого будет выполнена команда;
* **-t** - связать tty с контейнером для работы ввода и вывода;
* **-v** - примонтировать директорию основной системы в контейнер;
* **-d** – запуск контейнера в фоновом режиме.

На изображении 27 представлена проверка успешной интеграции Docker’a в OC Ubuntu с помощью команды *docker -v*.

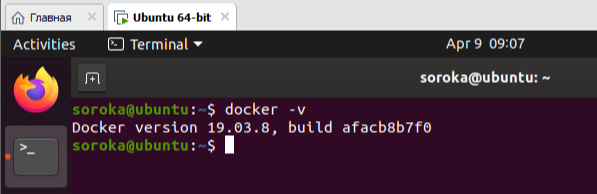


Рисунок 27 – Проверка Docker’а после установки

**Работа с Docker’ом**

Следующим шагом после успешной установки Docker’а в операционную систему, можно начинать взаимодействие с данным средством виртуализации. Для этого, необходимо опредлелиться с образом операционной системы, который указан в задании по варианту. В данном случае, все производимые манипуляции с Dorcker’ом, будут описаны на примере работы с OC Ubuntu.

Для того, чтобы вывести список доступных образов операционных систем в Docker’е используется команда *docker search <наименование искомой операционной системы>.* Как показано на изображении 28, при использовании данной команды, в консоли появится список имеющихся образов с операционной системой, название которой было задано в поиске. Из данного списка необходимо выбрать официальный (чистый, без внесённых дополнений) образ. Определить данную конфигурацию возможно по столбику с названием “Official” и отметкой [OK] в нём.

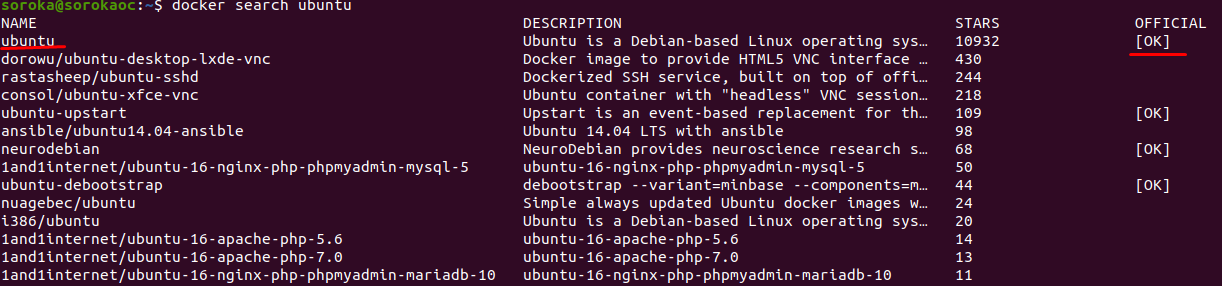


Рисунок 28 – Поиск необходимого образ

Далее рассмотрим листинг кода для работы с Docker файлом c последующем его описанием:

FROM Ubuntu

RUN apt update **&&** apt **install** **git** -y **&&** apt **install** build-essential -y **&&** **apt-get install** gcc-multilib -y

WORKDIR **/**home**/**app**/**

RUN **git clone** https:**//**github.com**/**Soroka**/**SP\_laba.git \

**&&** **cd** SP\_laba**/** \

**&&** **gcc** -fno-pie -no-pie laba.cpp -o lab -g -lstdc++

WORKDIR **/**home**/**app**/**SP\_laba**/**

После того, как необходимый образ операционной системы был найден, необходимо использовать команду *FROM <наименование операционной систем>),* к примеру *FROM Ubuntu*, как представлено на рисунке 29 списке. Если существует необходимость в установке образа операционной системы определённой версии, то нужно через знак двоеточия, который ставится после названия OC, указать нужную версию.

Cледующая строка, описанная в листинге, начинающаяся с команды *RUN* (выполнение команды в контейнере), может быть написана в трёх вариациях:

* + - * использование между выполнением команд логического ИЛИ (||). Тогда в данном случае каждая команда из строчки будет выполняться независимо от результата выполнения предыдущей команды (команды выполняются параллельно);
      * использование между выполнением команд логического И (&&). В таком случае, каждая следующая команда будет выполнена в случае успешного выполнения предыдущей (команды выполняются последовательно);
      * использование раздельного написания каждой команды (одна команда – одна строчка).

Далее идут команды *apt update* и *apt install git -y.* Рассмотрим ее составные части:

* + - * *apt* - (Advanced Package Tool) — инструмент командной строки для взаимодействия с системой управления пакетами (менеджер пакетов);
      * *update –* команда менеджера пакетов, позволяющая произвести обновление всех установленных пакетов системы;
      * *install* – команда менеджера пакетов, позволяющая установить новый пакет, в данном случаем это «git»;
      * *-y* – ключ (флаг подтверждения), сокращение от слова «YES».

Затем в строке следует команда *apt install build-essential,* это пакет, который содержит ссылки на множество пакетов, необходимых для создания программного обеспечения в целом (устанавливает необходимые пакеты для работы с gcc/g++).

Далее следует команда *apt-get install gcc-multilib* – команда устанавливающая компилятор g++/gcc.

*WORKDIR –* команда, с помощью которой создается рабочая директория;

*RUN git clone* – команда, выполняющая клонирование с git-репозитория;

*cd* – перейти в директорию;

*gcc -fno-pie -no-pie laba.cpp -o lab -g -lstdc++ -* позволяет скомпилировать исполняемый файл из исходного файла (cpp с ассемблерной вставкой).

Также можно использовать команду *COPY <из директории> <в директорию>* для того, чтобы скопировать файл с виртуальной машины (в данном случае) в Docker контейнер.

Команда *CMD* — инструкция для запуска некоторого исполняемого файла при запуске самого контейнера.

Пара моментов о *CMD*:

Только одна *CMD*-инструкция на весь Docker-файл. Иначе все кроме последней будут проигнорированы;

*CMD* может включать исполняемый файл;

Если же *CMD* не содержит никакого файла, обязательно должна быть инструкция *ENTRYPOINT*. В этом случает обе инструкции должны быть в формате JSON;

Аргументы командной строки для запуска Docker переопределяют аргументы, предоставленные *CMD* в Docker-файле[3][4].

**Установка дополнительного программного обеспечения**

После успешной установки операционной системы, необходимо установить дополнительное программное обеспечения, для дальнейшего выполнения лабораторной работы.

В первую очередь, необходимо установить текстовый редактор (Sublime Text, Vim или др.). Установить данное программное обеспечение можно установить, как из «Магазина приложений», так и через терминал. Через терминал в Unix – подобных система, можно установить практически любое приложение, если соблюдается условие того, что для данная программа есть в репозитории.

В ходе выполнения, данной лабораторной работы, будет использоваться текстовый редактор Sublime Text. Sublime Text— проприетарный текстовый редактор, поддерживает большое количество языков программирования и имеет возможность подсветки синтаксиса для C, C++, C#, CSS, D, Dylan, Erlang, HTML, Groovy, Haskell, Java, JavaScript, LaTeX, Lisp, Lua, Markdown, MATLAB, OCaml, Perl, PHP, Python, R, Ruby, SQL, TCL и XML. Установка данной программы представлена на изображениях 29, 30 [10].

Для установки требуется выполнить команду *sudo apt install sublime-text*, если у вас в качестве операционной системы установлена Ubuntu.

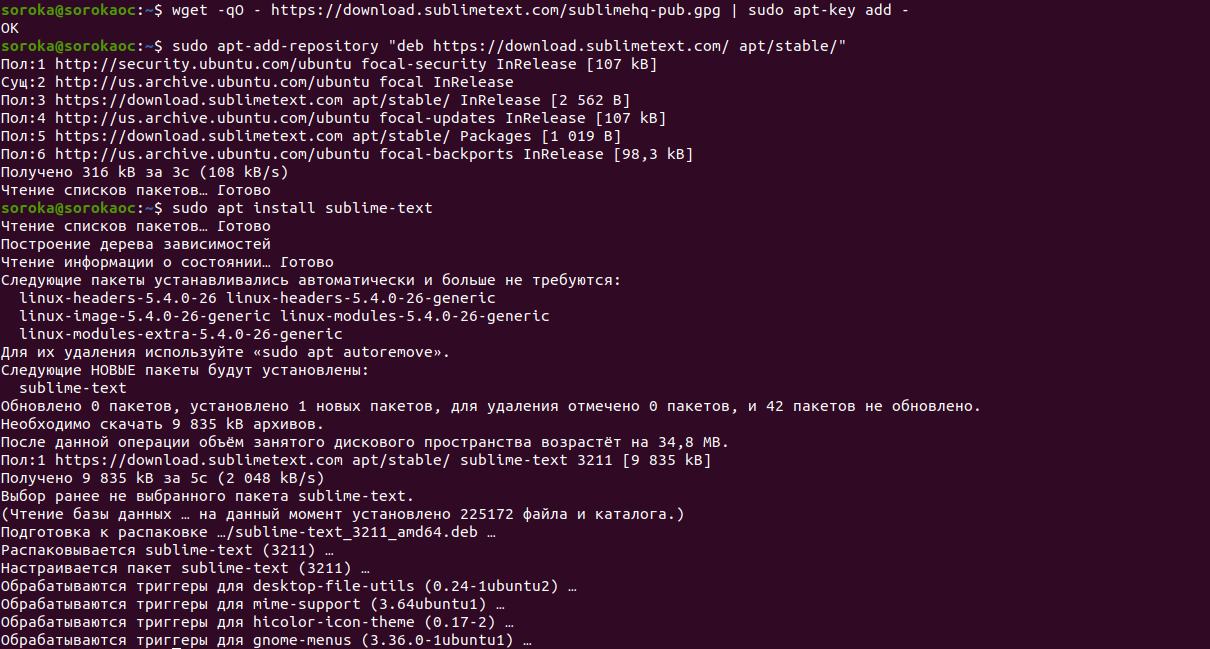


Рисунок 29 – Установка Sublime Text через терминал

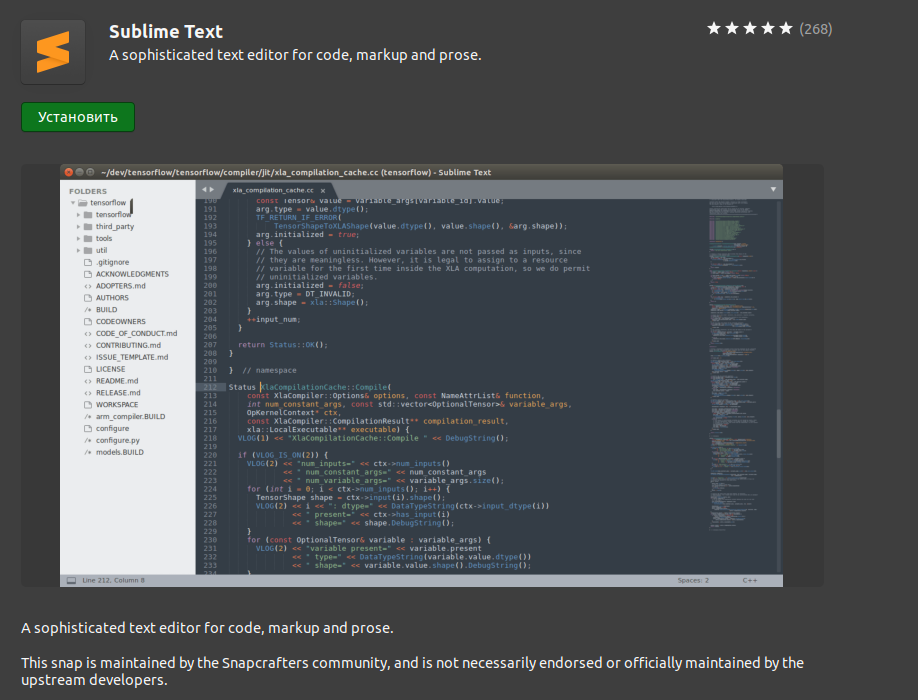


Рисунок 30 – Установка Sublime Text через магазин приложений

Следующим шагом, необходимо установить GDB. GDB - переносимый отладчик проекта GNU, который работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования, включая Си, C++, Free Pascal, FreeBASIC, Ada, Фортран, Python3, Swift, NASM и Rust. Проще говоря, GDB – это переносимый отладчик исполняемых файлов, который позволяет производить отладку на уровне машинного кода. Установка GDB, реализуется с помощью двух команд в терминале(*sudo apt update* и *sudo apt install gdb*), как показано на рисунке 31. Ознакомиться с основными командами GDB, возможно в методическом пособии «Знакомство с ассемблером», лабораторная работа №2, стр. 30 – 32.

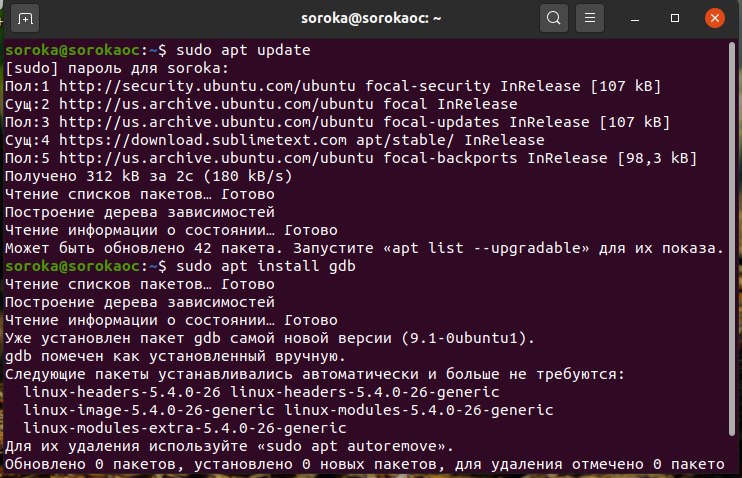


Рисунок 31 – Установка gdb

**Ассемблерная вставка**

В программировании, ассемблерной вставкой называют возможность компилятора встраивать низкоуровневый код, написанный на ассемблере, в программу, написанную на языке высокого уровня, например, Си и С++. Использование ассемблерных вставок может преследовать следующие цели:

- оптимизация: С этой целью, вручную пишется ассемблерный код, реализующий наиболее критичные в отношении производительности части алгоритма. Это позволяет программисту использовать в полной мере свою изобретательность, не ограничиваться конструкциями компилятора;

- доступ к специфичным инструкциям процессора: Некоторые процессоры поддерживают специальные инструкции, такие как сравнение с обменом и test-and-set — инструкции, которые могут быть использованы для реализации семафоров или других примитивов синхронизации и блокировок. Практически все современные процессоры имеют такие или сходные инструкции, так как они необходимы для реализации многозадачности. Специальные инструкции можно найти в системах команд следующих процессоров: SPARC VIS, Intel MMX и SSE, Motorola AltiVec;

- системные вызовы: Языки программирования высокого уровня редко предоставляют прямую возможность делать системные вызовы, для этих целей используется ассемблерный код.

Для того, чтобы подключить модуль ассемблера в Visual Studio, его необходимо предварительно откомпилировать. Затем полученный объектный модуль, в котором находится ассемблерная процедура, необходимо подключить к приложению в файл проекта следующим образом:

*extern “C” void \_\_<конвенция> <название процедуры> (<тип данных> <переменная>);*

Модуль на Ассемблере необходимо транслировать с опциями:

* + - * *ML /c /coff program.asm*
      * *tasm /ml program.asm*

Ниже приведено схематическое описание конвенций, на котором видно каким образом передаются данные (сколько через регистр, сколько через стек), на кого возложена функция очистки стека.

*; F(A,B,C,D)*

\_cdecl: *; прямой порядок расположения данных на стеке*

*; обратный по времени способ запихивания*

*; допускает переменное число параметров*

*; поскольку стек очищает тот, кто вызвал функцию*

**push** D

**push** C

**push** B

**push** A

**call** F

*; ret*

**inc** **esp**, 16 *; очистка стека здесь*

\_stdcall: *; порядок данных - тот же самый*

*; но стек очищает функция*

*; поэтому число параметров фиксировано*

**push** D

**push** C

**push** B

**push** A

**call** F

*; ret 16 ; очистка стека - в функции*

\_fastcall: *; аналогично stdcall, но два первых параметра передаются через регистры*

**push** D

**push** C

**mov** **edx**, B

**mov** **eax**, A

**call** F

*; ret 8 ; очистка стека - в функции*

\_pascal: *; обратный порядок данных на стеке, прямой порядок запихивания*

*; поэтому число параметров фиксировано, и следовательно, стек очищает функция*

**push** A

**push** B

**push** C

**push** D

**call** F

*; ret 16*

Где F – возвращаемое значение функции, A,B,С,D – передаваемые переменные.

В случае использования компилятора *GCC* ассемблерные команды нужно брать в двойные кавычки. При этом в конце каждой команды добавлять суффикс, состоящий из символов переноса строки и табуляции ‘\n’ ‘\t’. Для оформления вставки можно использовать ключевые слова asm или \_\_asm\_\_. Формат ассемблерной вставки для *GCC* в общем случае выглядит так:

\_asm\_(assembler template */\* ассемблерная вставка\*/*

: output operands */\* выходные операнды \*/*

: input operands */\* входные операнды \*/*

: list of clobbered registers */\* разрушаемые регистры \*/*

);

 ы

**OutputOperands**

OutputOperands связывают имя, видимое из ассемблерной вставки, с l-value кода на С++.

Их общий вид:

[ [asmSymbolicName] ] constraint (cvariablename)

asmSymbolicName - имя видимое из ассемблерной вставки (если OutputOperand указан как [foo]"m"(bar), то в ассемблерной вставке имя будет доступно как %[foo]). Эта часть не обязательна, если имя опущено, то операнд доступен по номеру (%0, %1 и т. д.). Компилятор проводит макроподстановку и вставляет в результирующий ассемблерный файл конкретное выражение (известное ему), обозначающее данное l-value. См. пример ниже.

constraint - ограничения на тип операнда (область памяти, регистр, возможность записи/чтения). Подробнее о constraint см. ниже.

cvariablename - выражение на C++, обозначающее l-value, связанное с данным именем (не обязательно имя переменной).

**InputOperands**

InputOperands связывают имя, видимое из ассемблерной вставки, со значением выражения на С++.

Их общий вид:

[ [asmSymbolicName] ] constraint (cexpression)

asmSymbolicName - имя видимое из ассемблерной вставки (подробности см. выше).

constraint - ограничения на тип операнда.

cexpression - выражение на C++, связанное с данным именем.

В таблице 3 представлен список входных и выходных операндов.

Таблица 3 – Выходные и входные операнды

|  |  |
| --- | --- |
| **Constraint** | **Значение** |
| **Общие** |  |
| m | Адресуемая область памяти |
| r | Регистр общего назначения |
| i | Непосредственный константный операнд |
| g | Регистр общего назначения, область памяти или непосредственный операнд |
| X | Любой операнд |
| 0, 1, 2, ... 9 | Операнд, совпадающий с операндом с указанным индексом (*matching constraint*) |
| **x86/x86\_64** |  |
| a, b, c, d | Регистры ax, bx, cx, dx и их версии (ah,ecx,rdx) |
| S, D | Регистры si, di и их версии |
| A | Пара ax:dx и её версии |
| f | Регистр стека FPU |
| t | Верхний регистр стека FPU (ST(0)) |
| u | ST(1) |
| y | Любой регистр MMX |
| x | Любой регистр SSE |
| Yz | Регистр xmm0 |
| G | Вещественная константа x387 |
| C | Нулевая константа SSE |

Шаблон ассемблера (assembler template) состоит из инструкций ассемблера. Двоеточие отделяет шаблон ассемблера от первого выходного операнда, а другие разделители (тоже двоеточие), отделяют последний выходной операнд от первого входного, если таковые имеются. При этом, если выходных операндов нет, но есть входные операнды, то необходимо поместить подряд два двоеточия там, где должны быть выходные операнды.

Директива *.intel\_syntax* меняет синтаксис AT&T на синтаксис Intel; необходимо дополнительно указывать смену синтаксиса для операндов инструкций,*noprefix*, что позволит писать код в более близком к диалекту nasm виде, не используя при записи имен регистров префикс *%* [2][5][6].

**Примеры работы с ассемблерной вставкой**

Рассмотрим пример работы программы сложения двух чисел на языке С++ с использованием ассемблерной вставки. Код написан и откомпилирован в среде разработки Visual Studio 2019. Результат работы программы представлен на изображении 32.

Листинг кода на С++ с асcемблерной вставкой:

#include <iostream>

using namespace std;

int main()

{

//Сложение двух чисел;

setlocale(LC\_ALL, "");

int a, b, Sum;

cout << "Введите число a:**\n**";

cin >> a; // Ввод числа "a"

cout << "Введите число b:**\n**";

cin >> b; // Ввод числа "b"

\_asm

{

mov Sum, 0 // Присвоение переменной Sum значения 0

mov eax, a // Присвоение регистру eax значение "a"

mov ebx, b // Присвоение регистру ebx значение "b"

add eax, ebx // Прибавление значение регистра ebx к значению регистра eax

add Sum, eax // Прибавление значение регистра eax к значению переменной Sum

}

cout << "Результат сложения:" << Sum << endl; // Вывод результата суммирования

system("pause");

return 0;

}

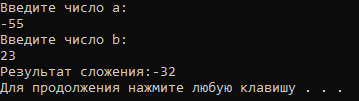


Рисунок 32 – Результат работы программы

Следующем примером необходимо написать программу, в которой создается числовой массив и для данного массива вычисляется сумма квадратов элементов массива. На рисунке 33 представлен результат работы написанной программы. Код написан в программе Visual Studio 2019.

Листинг кода на С++ с асcемблерной вставкой:

#include <iostream>

using namespace std;

int main()

{

int A[10], Sum;

setlocale(LC\_ALL, "");

cout << "Введите 10 чисел через пробел:";

for (int i = 0; i < 10; i++) cin >> A[i]; // Ввод значений массива

\_asm

**{**

**mov** Sum, 0 // Инициализация суммы квадратов

**cld** // Сканирование элементов

**lea** **esi**, A // Постановка указателя на начало массива

**mov** **ecx**, 10 // Заданный размер массива

M:**lodsd** // Цикл суммирования, загрузка следующего элемента

**imuL** **eax** // Возведения элемента массива в квадрат

**add** Sum, **eax** // Суммирование элементов возведённых квадрат

**loop** M

**}**

cout << "Сумма квадратов:" << Sum << endl; // Вывод результата

system("pause");

return 0;

}



Рисунок 33 – Результат работы программы

Далее рассмотрим пример работы программы, выполняющей транспонирование матрицы (двумерного массива). На высокоуровневом языке программирования C++ напишем программу, которая позволяет выполнять транспонирование случайно сгенерированного двумерного массива. Следующий код написан в линуксе.

Листинг программы на C++ (без вставки):

#include <iostream>

#include <time.h>

using namespace std;

int main()

{

srand((unsigned int)time(0));

long long n=rand()%8+2, mass[n][n],i,j;

for(i=0;i<n;i++){

for(j=0;j<n;j++){

mass[i][j]=rand()%89+10;

}

}

cout<<"Исходная матрица:"<<endl;

for(i=0;i<n;i++){

for(j=0;j<n;j++){

cout<<mass[i][j]<<" ";

}

cout<<endl;

}

//транспонирование

for(i=0;i<n;i++){

for(j=i;j<n;j++){

unsigned int tmp=mass[i][j];

mass[i][j]=mass[j][i];

mass[j][i]=tmp;

}

}

cout<<"**\n**Транспонированная матрица:"<<endl;

for(i=0;i<n;i++){

for(j=0;j<n;j++){

cout<<mass[i][j]<<" ";

}

cout<<endl;

}

return 0;

}

Выполним программу запуском следующей команды, предварительно перейдя в директорию, в которой находится исходный модуль программы. Процесс компилирования программы, представлен на рисунке 34.

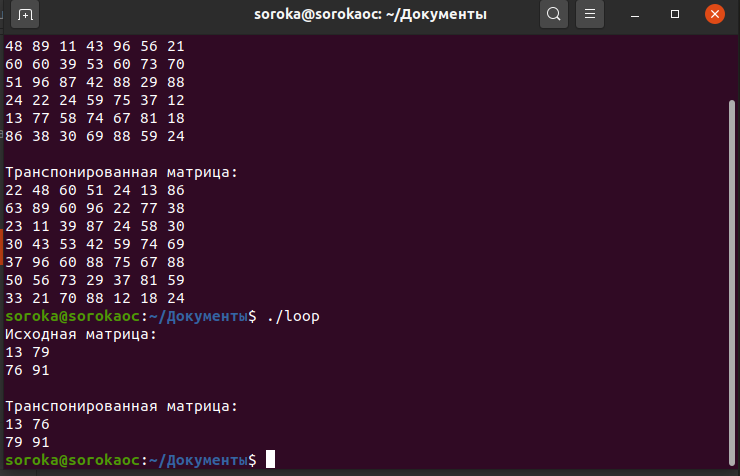


Рисунок 34 – Выполнение программы

После написания программы на языке С++, заменим часть кода, отвечающего за транспонирование матрицы на ассемблерную вставку, и также проверим работоспособность, как показано на изображении 35.

Листинг кода C++ с ассемблерной вставкой:

#include <iostream>

#include <time.h>

using namespace std;

int main()

{

srand((unsigned int)time(0));

long long n=rand()%8+2, mass[n][n],i,j;

for(i=0;i<n;i++){

for(j=0;j<n;j++){

mass[i][j]=rand()%89+10;

}

}

cout<<"Исходная матрица:"<<endl;

for(i=0;i<n;i++){

for(j=0;j<n;j++){

cout<<mass[i][j]<<" ";

}

cout<<endl;

}

long long \*ptr\_array = &mass[0][0], \*end\_array = &mass[n-1][n-1]; // Указатель на массив, чтобы ассемблерная вставка к нему обращалась

\_\_asm\_\_(

"mov $8, %%rax**\n\t**" // Поместили 8 в rax

"mulq %[n]**\n\t**" // Умножили rax на n (получили сдвиг для перехода по строкам)

"mov %[ptr\_array], %%rbx**\n\t**" // Поместили указатель на начало матрицы в rbx

"mov %%rbx, %%rcx**\n\t**" // Поместили указатель на начало матрицы в rcx

"add %%rax, %%rcx**\n\t**" // Прибавили к rcx содержимое rax (сдвинули указатель вниз на одну строку)

"add $8, %%rcx**\n\t**" // Прибавили к rcx 8 (сдвинули указатель вправо на один столбец)

"push %%rcx**\n\t**" // Поместили в стек rcx (адрес следующего элемента главной диагонали)

"push %%rax**\n\t**" // Поместили в стек rax (сдвиг для перехода по строкам)

"mov %[n], %%rcx**\n\t**" // Поместили в rcx значение n

"dec %%rcx**\n\t**" // Уменьшили на 1 значение rcx (получили в rcx n-1)

"mulq %%rcx**\n\t**" // Умножили rax (сдвиг для перехода по строкам) на rcx (n-1)

"add %%rax, %[ptr\_array]**\n\t**" // Прибавили к ptr\_array содержимое rax (получили указатель на первый элемент последней строки матрицы)

"mov %%rbx, %%rcx**\n\t**" // Вернули в rcx указатель на начало матрицы

"loop\_start:**\n\t**"

"add $8, %%rbx**\n\t**" // Увеличили адрес в rbx на 8 (переместились на один элемент вправо)

"pop %%rax**\n\t**" // Взяли в rax из стека смещение на следующую строку

"add %%rax, %%rcx**\n\t**" // Увеличили значение в rcx на смещение на следующую строку (переместились вниз на один элемент)

"push %%rax**\n\t**" // Вернули в стек смещение на следующую строку

"push (%%rbx)**\n\t**" // Обмен значений элементов матрицы через стек

"push (%%rcx)**\n\t**"

"pop (%%rbx)**\n\t**"

"pop (%%rcx)**\n\t**"

"cmp %%rcx, %[ptr\_array]**\n\t**" // Проверили что не достугнута последняя строка матрицы

"je next\_iteration**\n\t**" // Если достигнута - переход на next\_iteration

"jmp loop\_start**\n\t**" // Если НЕ достигнута - переход на start\_loop

"next\_iteration:**\n\t**"

"add $8, %[ptr\_array]**\n\t**" // Сдвинули указатель на один элемент последней строки матрицы вправо

"mov %[ptr\_array], %%rax**\n\t**" // Переместили указатель в rax

"cmp %%rax, %[end\_array]**\n\t**" // Проверили что достигнут конец матрицы

"je end\_asm**\n\t**" // Если достигнут - завершение, иначе продолжение

"pop %%rax**\n\t**" // Взяли в rax из стека смещение на следующую строку

"pop %%rbx**\n\t**" // Взяли в rbx из стека указатель на следующий элемент главной диагонали, он станет началом новой "усеченной" матрицы

"mov %%rbx, %%rcx**\n\t**" // Поместили в rcx указатель на следующий элемент главной диагонали

"add %%rax, %%rcx**\n\t**" // Сдвинули указатель в rcx на одну строку вниз

"add $8, %%rcx**\n\t**" // Сдвинули указатель в rcx на один элемент вправо

"push %%rcx**\n\t**" // Вернули rcx в стек

"mov %%rbx, %%rcx**\n\t**" // Поместили в rcx значение из rbx

"push %%rax**\n\t**" // Вернули rax в стек

"jmp loop\_start**\n\t**" // Переход на loop\_start

"end\_asm:**\n\t**"

:

:[ptr\_array]"M"(ptr\_array), [end\_array]"M"(end\_array), [n]"M"(n)

:"%rax", "%rbx", "%rcx"

);

cout<<"**\n**Транспонированная матрица:"<<endl;

for(i=0;i<n;i++){

for(j=0;j<n;j++){

cout<<mass[i][j]<<" ";

}

cout<<endl;

}

return 0;

}

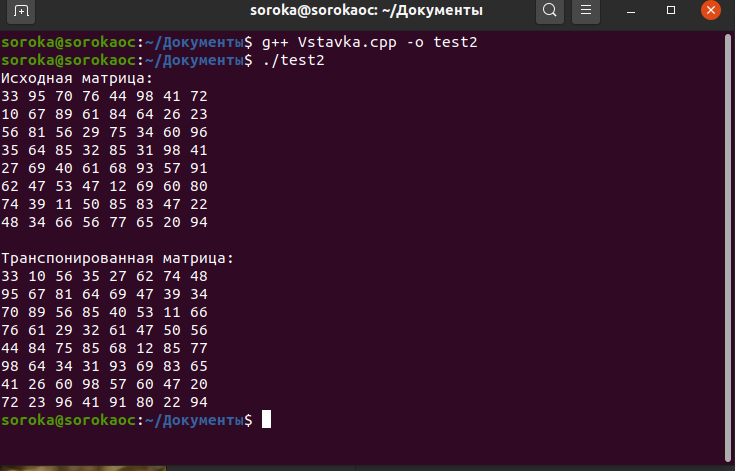


Рисунок 35 – Проверка работоспособности конечной программы

**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. Получить индивидуальное задание у преподавателя. Решение основной задачи реализовать на ассемблере в виде процедуры. Далее получить объектный модуль. Вызывать из кода на C++ процедуру из объектного файла и скомбинировать их в один исполняемый файл. Альтернативный вариант - использовать ассемблерные вставки.

4. Написать отчет и защитить у преподавателя. Залить код, докерфайл в git, на мудл залить архив с отчетом и всеми исходными кодами, dockerfile.

**Варианты индивидуальных заданий**

1. Напишите программу, в которой создается два числовых массива одинакового размера. Необходимо вычислить сумму попарных произведений элементов этих массивов. Так, если через ak и bk, обозначить элементы массивов (индекс 0 ≤ k< n), то необходимо вычислить сумму akbk

2. Напишите программу, в которой создается числовой массив и для этого массива вычисляется сумма квадратов элементов массива.

3. Напишите программу, в которой создается двумерный числовой массив и для этого массива вычисляется сумма квадратов его элементов.

4. Напишите программу, в которой создается квадратная матрица (реализуется через двумерный массив). Матрицу необходимо заполнить числовыми значениями (способ заполнения выбрать самостоятельно), а затем выполнить транспонирование матрицы: матрица симметрично «отражается» относительно главной диагонали, в результате чего элемент матрицы аij становится элементом аji и наоборот.

5. Напишите программу, в которой создается квадратная матрица (реализуется через двумерный массив). Матрица заполняется случайными числами, после чего выполняется «поворот по часовой стрелке»: первый столбец становится первой строкой, второй столбец становится второй строкой, и так далее.

6. Напишите программу, в которой для двумерной квадратной числовой матрицы вычисляется след — сумма диагональных элементов матрицы.

7. Напишите программу, в которой для одномерного числового массива вычисляется наибольший (наименьший) элемент и позиция, на которой элемент находится в массиве.

8. Напишите программу, в которой создается одномерный числовой массив. После заполнения значениями (например, случайными числами) массива в нем нужно выполнить циклическую перестановку элементов. Количество позиций для циклической перестановки вводится пользователем с клавиатуры.

9. Напишите программу, в которой создается и заполняется натуральными числами двумерный массив. Заполнение начинается с левого верхнего элемента слева направо, сверху вниз (то есть заполняется сначала первая строка, затем вторая, и так далее).

10. Напишите программу, в которой создается и заполняется натуральными числами двумерный массив. Заполнение начинается с левого верхнего элемента сверху вниз, слева направо (то есть заполняется сначала первый столбец, затем второй и так далее).

11. Напишите программу, в которой создается квадратная матрица. Элементам на обеих диагоналях присваиваются единичные значения, все остальные элементы нулевые.

12. Напишите программу для вычисления произведения прямоугольных матриц (количество строк и столбцов в матрицах различное). Перемножаются матрицы A (размерами m на n) и B (размерами n на l). Результатом является матрица C (размерами m на l). Рабочая формула для вычисления значений матрицы C имеет вид, где 1 ≤ i ≤ m и 1≤ j ≤ l.

13. Напишите программу, в которой создается символьный массив для записи текста. В массив записывается текст, после чего необходимо изменить порядок следования символов в тексте на противоположный: последний символ становится первым, предпоследний символ становится вторым и так далее.

14. Напишите программу, в которой создается двумерный числовой массив и заполняется случайными числами. Для каждой строки (или столбца) двумерного массива определяется наибольший (или наименьший) элемент, и из таких элементов создается одномерный числовой массив.

15. Напишите программу, в которой создается символьный массив для записи текста. Подсчитать для записанного в массиве текста количество слов и длину каждого слова. Словами считать набор символов между пробелами.

16. Напишите программу, в которой создается динамический массив. Размер массива — случайное число (то есть генерируется случайное число, которое определяет размер массива). Заполнить массив «симметричными» значениями: первый и последний элемент получают значение 1, второй и предпоследний элемент получают значение2, и так далее.

17. Напишите программу, в которой создается три одномерных числовых массива одинакового размера. Первые два массива заполняются случайными числами. Третий массив заполняется так: сравниваются элементы с такими же индексами в первом и втором массиве и в третий массив на такую же позицию записывается большее (или меньшее) из сравниваемых значений.

18. Напишите программу, в которой создается два динамических массива (разной длины). Массивы заполняются случайными числами. Затем создается третий динамический массив, и его размер равен сумме размеров первых двух массивов. Третий массив заполняется так: сначала в него записываются значения элементов первого массива, а затем значения элементов второго массива.

19. Напишите программу, в которой создается два динамических массива одинакового размера. Массивы заполняются случайными числами. Затем создается третий динамический массив, и его размер в два раза больше размера каждого из первых двух массивов. Третий массив заполняется поочередной записью элементов из первых двух массивов: сначала записывается значение элемента первого массива, затем значение элемента второго массива, затем снова первого и снова второго, и так далее.

20. Напишите программу, в которой создается двумерный числовой массив. Массив заполняется случайными числами. На его основе создается новый массив, который получается «вычеркиванием» из старого одной строки и одного столбца. Номера «вычеркиваемых» столбца и строки определяются вводом с клавиатуры или с помощью генератора случайных чисел.

**Список литературы:**

1. МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ НА АССЕМБЛЕРЕ. СВЯЗЬ РАЗНОЯЗЫКОВЫХ МОДУЛЕЙ - http://elearning.bmstu.ru/moodle/pluginfile.php/2978/mod\_data/content/784/2.%20Модульное%20программирование%20на%20ассемблере.%20Связь%20разноязыковых%20модулей.pdf
2. Ассемблерная вставка - <https://ru.wikipedia.org/wiki/Ассемблерная_вставка>
3. Установка Docker’a на Ubuntu - <https://losst.ru/ustanovka-docker-na-ubuntu-16-04>
4. Просто о виртуализации.Docker - [https://youtu.be/Sa7uOGczoHc](https://vk.com/away.php?utf=1&to=https%3A%2F%2Fyoutu.be%2FSa7uOGczoHc)
5. Что значат \_\_cdecl, \_\_fastcall, и \_\_stdcall специфики функций? - <https://www.rsdn.org/forum/cpp/159835.all>
6. Ассемблерные вставки в GCC - <https://www.gamedev.ru/code/articles/gcc_inline_asm?page=3>
7. Скачать Oracle Virtual Box - [https://www.virtualbox.org]
8. Скачать Vmware Workstation - [https://www.vmware.com/go/getworkstation-win]
9. Скачать операционной система Ubuntu - [https://ubuntu.ru/]
10. УСТАНОВКА SUBLIME TEXT 3 UBUNTU 18.04 - https://losst.ru/ustanovka-sublime-text-3-ubuntu-18-04