**Липецкий государственный технический университет**

Факультет автоматизации и информатики

Программа ДПО Intaro

ЛАБОРАТОРНАЯ РАБОТА №1

“Виртуализация, гипервизоры, создание виртуальных машин.

Средства разработки программного обеспечения в Linux”

Студент \_\_\_\_\_\_\_\_\_\_ Целищев А.Е.

ПМ-21-2

Руководитель \_\_\_\_\_\_\_\_\_\_ Кургасов В.В.

Доцент каф. АСУ

Липецк 2023 г

Оглавление

[Цели работы 3](#_Toc9114)

[Задание к лабораторной работе 3](#_Toc12389)

[Установка VM VirtualBox 4](#_Toc27089)

[Создание виртуальной машины, ее настройка 5](#_Toc4632)

[Установка гостевой ОС Linux (Ubuntu Server) 10](#_Toc32722)

[Настройка текстового редактора Vim, разработка на Linux 15](#_Toc5737)

[Выводы 20](#_Toc14593)

[Ответы на контрольные вопросы 21](#_Toc19725)

**Цели работы**

- Изучить состав инструментальных средств создания и сопровождения аппаратных конфигураций виртуальных машин (ВМ);

- Получить практические навыки выполнения типовых операций мониторинга и управления состояниями виртуальных машин;

- Изучить функциональные возможности интеграции виртуальных (гостевых) и физической (хостовой) машин;

- Ознакомиться с типами гипервизоров и их возможностями от различных разработчиков;

- Получить представление о технологиях и средствах разработки программного обеспечения в ОС Linux.

**Задание к лабораторной работе**

- Установить гипервизор, изучить понятия «виртуализация», «гипервизоры», изучить создание виртуальных машин;

- Установить 64-разрядную серверную ОС на выбор (Ubuntu, Debian);

- ЛР №3 из предложенного пособия, 1 задача по варианту (Вариант 6), ответы на контрольные вопросы.

**Установка VM VirtualBox**

Пройдем на официальный сайт <https://www.virtualbox.org> и в разделе *Downloads* увидим предлагаемые способы установки гипервизора VirtualBox. Так как лабораторная работа выполняется на ОС Ubuntu 22.04, имеются два варианта: скачать deb-пакет для соответствующей версии системы либо выполнить установку через терминал. Выберем последний способ.

Следуя инструкциям, добавим в файл /etc/apt/sources.txt требуемую строку:



Рис 1. Открываем файл при помощи текстового редактора nano

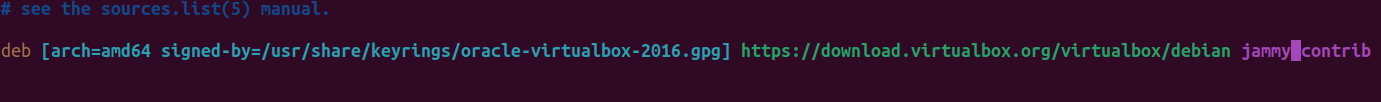


Рис 2. Вставка требуемой строки в файл /etc/apt/sources.txt

Установим и зарегистрируем публичный ключ Oracle, исполнив команду:

wget -O- https://www.virtualbox.org/download/oracle\_vbox\_2016.asc | sudo gpg --dearmor --yes --output /usr/share/keyrings/oracle-virtualbox-2016.gpg

Исполним следующую команду и дождемся установки VirtualBox:

sudo apt-get update && sudo apt-get install virtualbox-6.1

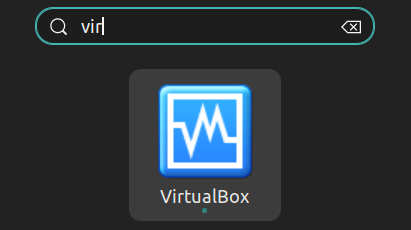


Рис 3. Успешно установленная программа.

**Создание виртуальной машины, ее настройка**

Для начала создадим виртуальную машину. Запустим VirtualBox, “Машина” -> “Создать”. Введем имя виртуальной машины, тип операционной системы и версию.

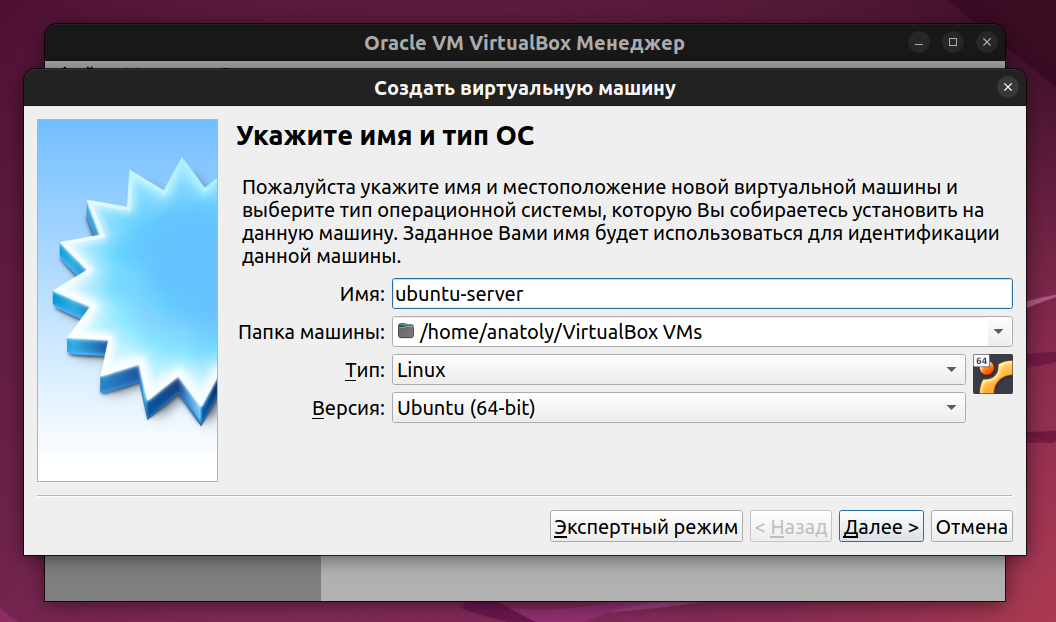


Рис 4. Указываем имя и тип ОС.

Выделим 2 гб оперативной памяти для ВМ.

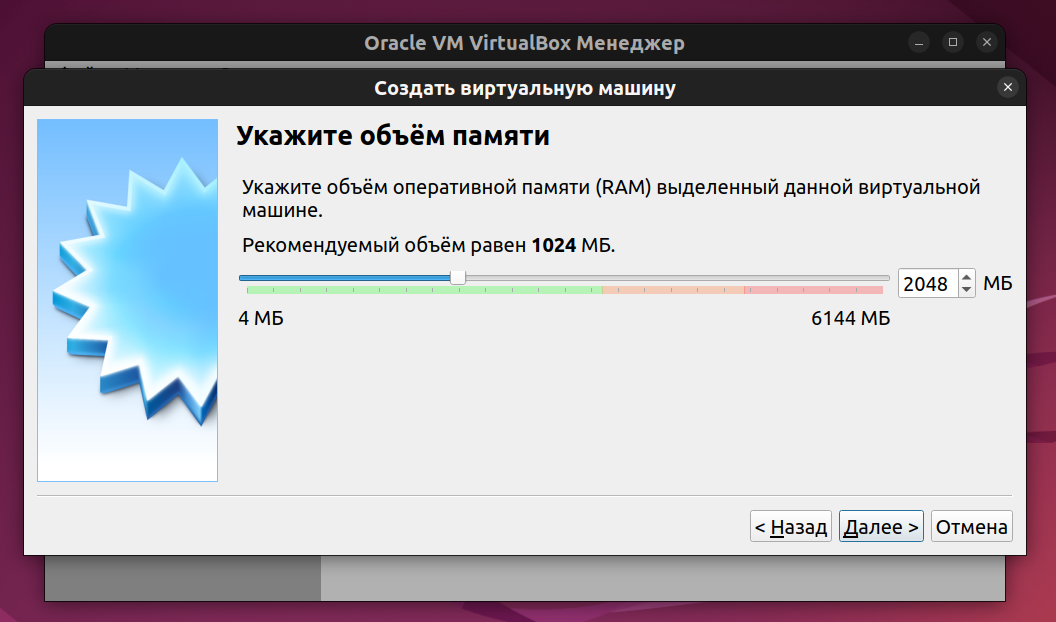


Рис 5. Выделение оперативной памяти объемом 2 гб.

Создадим новый виртуальный жесткий диск, тип VDI,

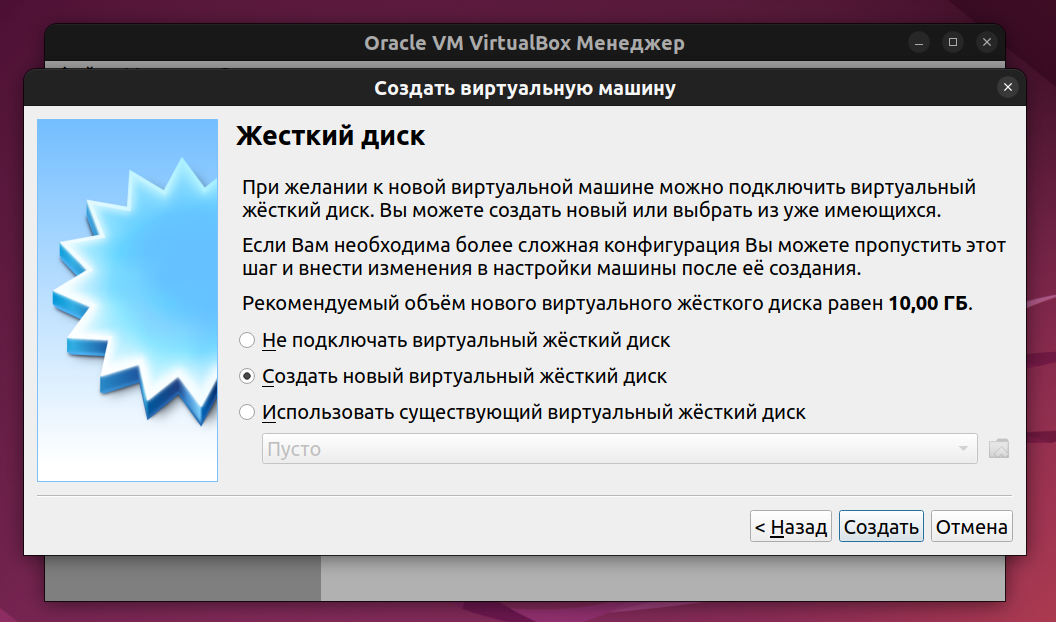


Рис 6. Создание нового виртуального жесткого диска.

Формат хранения - динамический. Хотя некоторые процессы впоследствии будут работать медленнее (такие как ввод/вывод данных), таким образом мы по возможности сэкономим память хостовой машины - в данном случае это важно, так как на хостовой машине, с которой выполняется работа, объем свободной памяти небольшой.

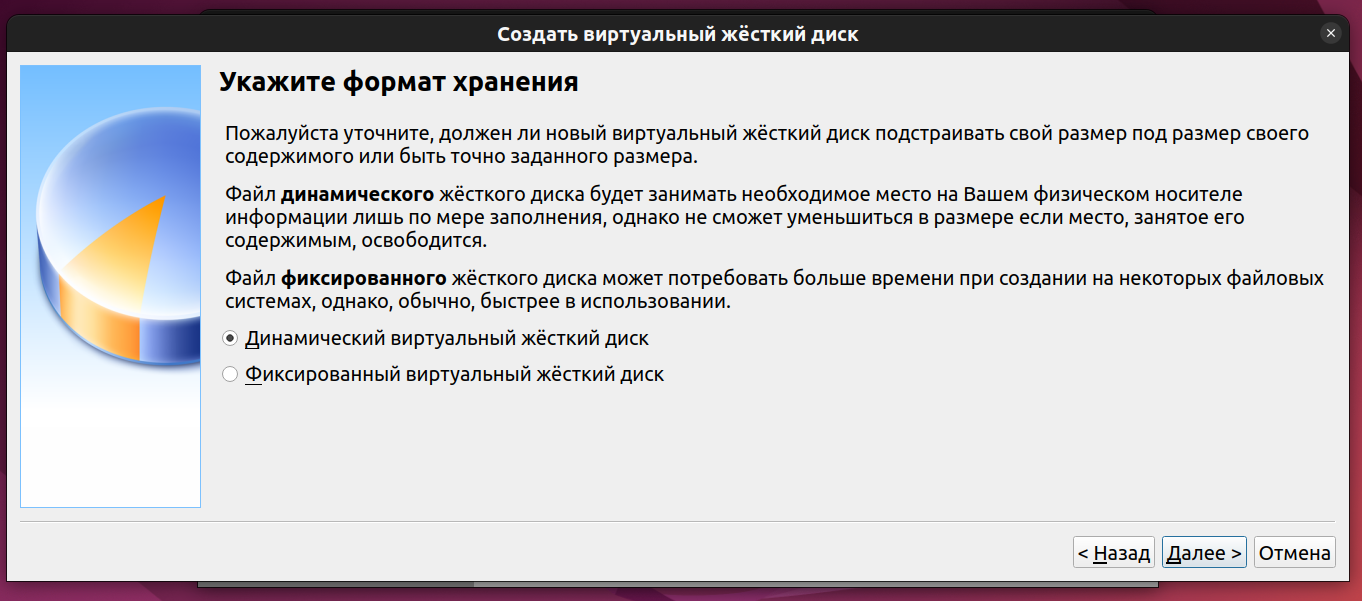


Рис 7. Выбираем динамический формат хранения.

Выделяем 6 гб для виртуального жесткого диска. (Впоследствии выяснилось, что такого объема памяти недостаточно, и была создана новая виртуальная машина на ОС Windows 10, где было больше памяти, и было выделено 18 гб с расчетом на то, что в данной виртуальной машине будет использоваться Docker)

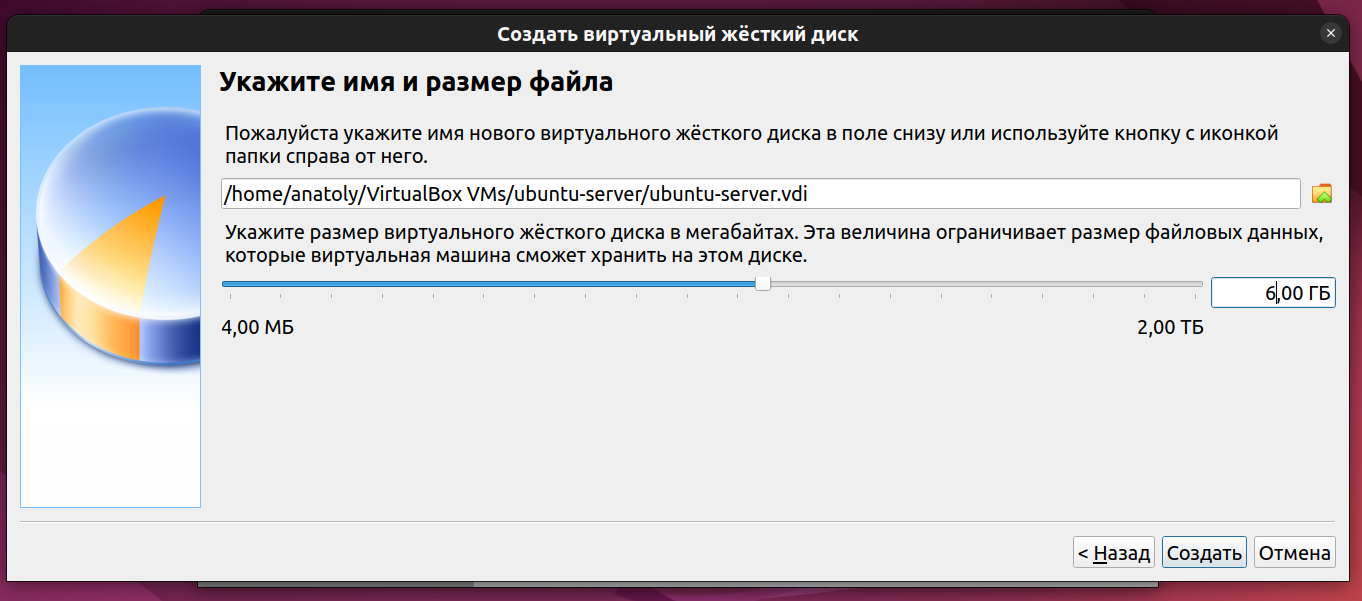


Рис 8. Обозначение максимально допустимого размера хранимых файловых данных.

Виртуальная машина создана и в данный момент выключена и не имеет конкретной операционной системы.

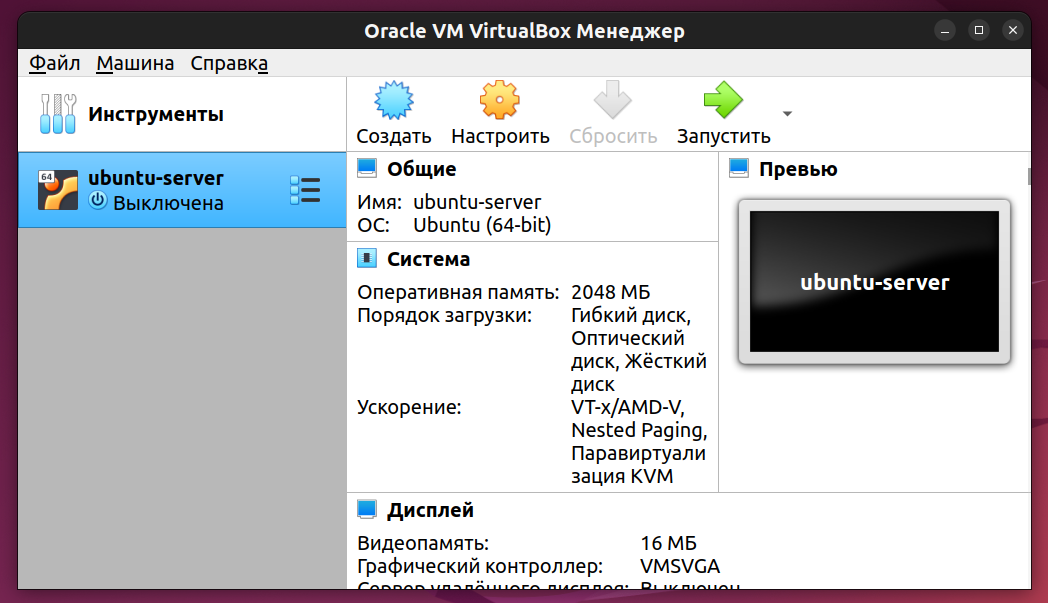


Рис 9. Результат создания виртуальной машины ubuntu-server.

Настроим процессор и видеопамять для более эффективной эмуляции гостевой машины. Выделим созданную машину и выберем пункт “Настроить”.

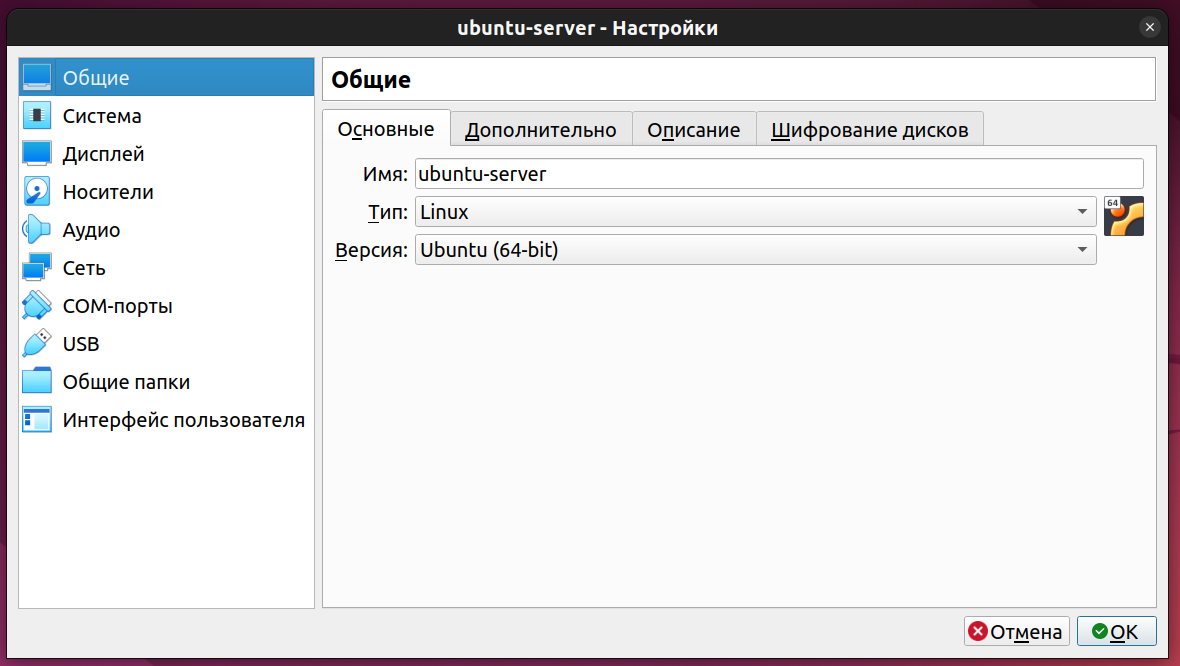


Рис 10. Окно настроек виртуальной машины ubuntu-server.

Перейдем в “Система” -> “Процессор”. Установим рекомендованное количество используемых ядер процессора - 2. Пусть предел загрузки ЦПУ будет 100%.

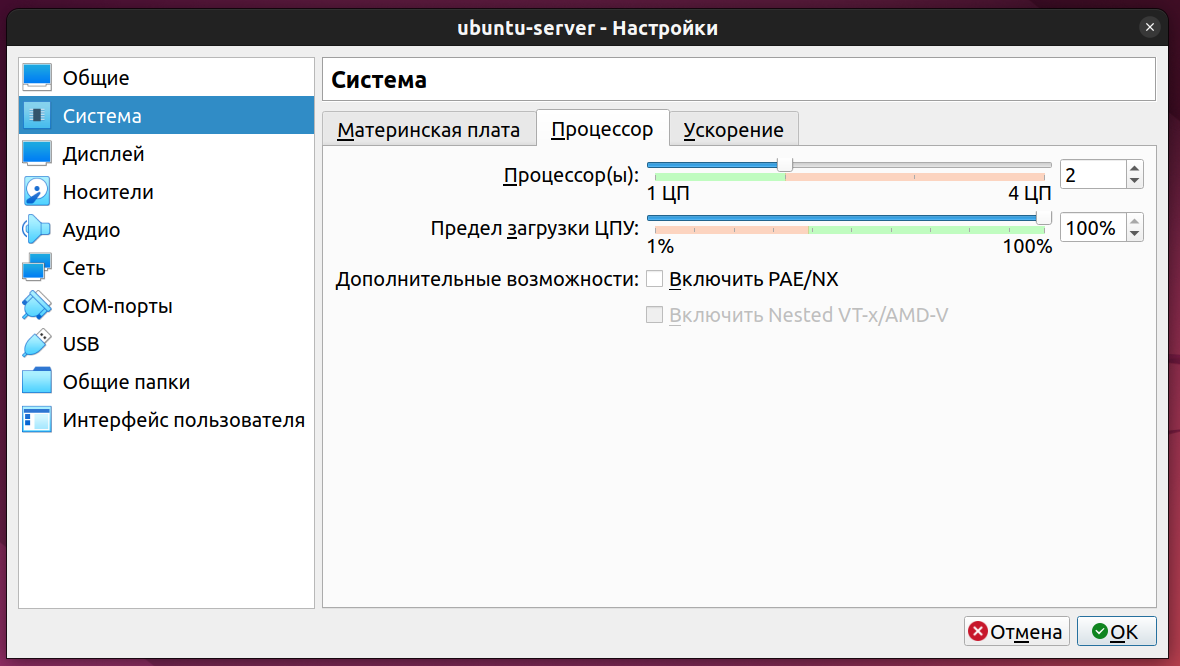


Рис 11. Настройка процессора для ВМ.

Во вкладке “Ускорение” включим пункт “Включить Nested Paging”

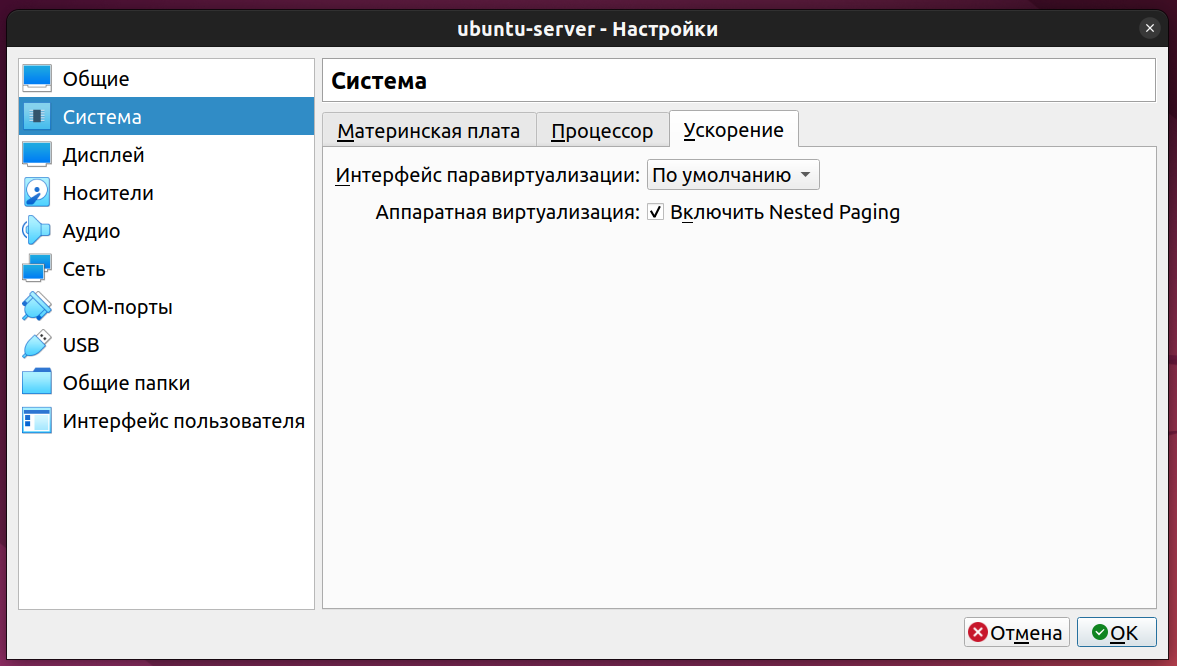


Рис 12. Включаем аппаратную виртуализацию.

Перейдем в “Дисплей” -> “Экран” и выделим 128 мб на видеопамять (вместо 18 мб по умолчанию).

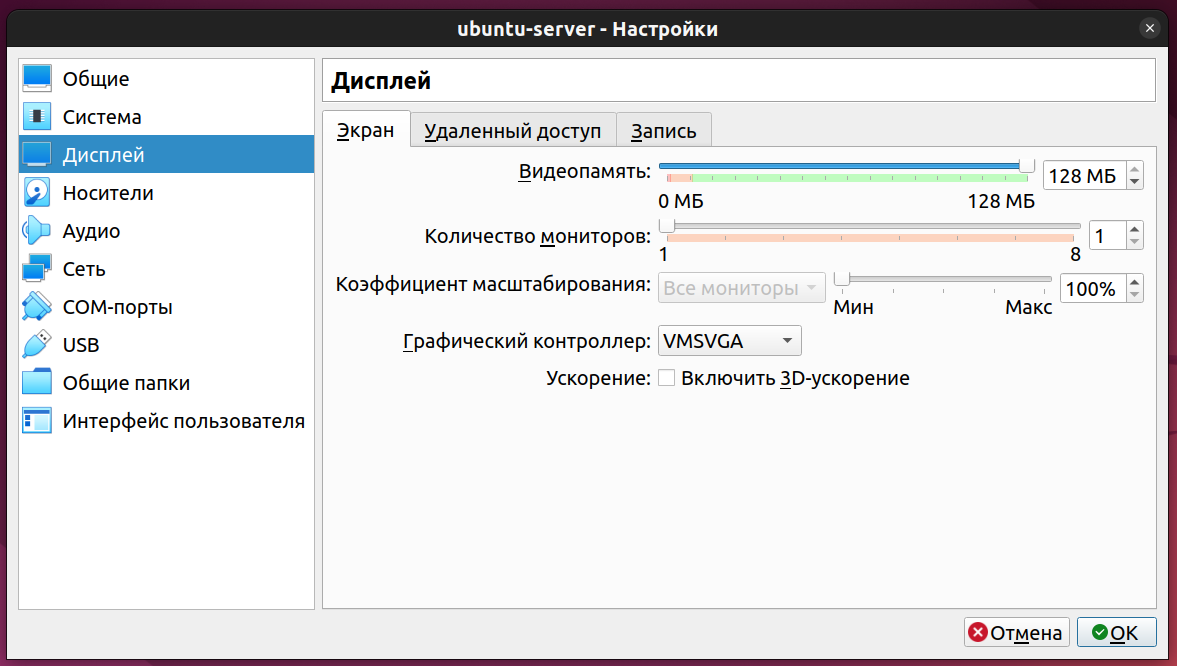


Рис 13. Выделение большего объема видеопамяти для ВМ.

**Установка гостевой ОС Linux (Ubuntu Server)**

Перейдем на официальный сайт дистрибутива, в раздел для скачивания Ubuntu Server <https://ubuntu.com/download/server> и скачаем iso-образ дистрибутива Ubuntu Server 22.04.3 LTS

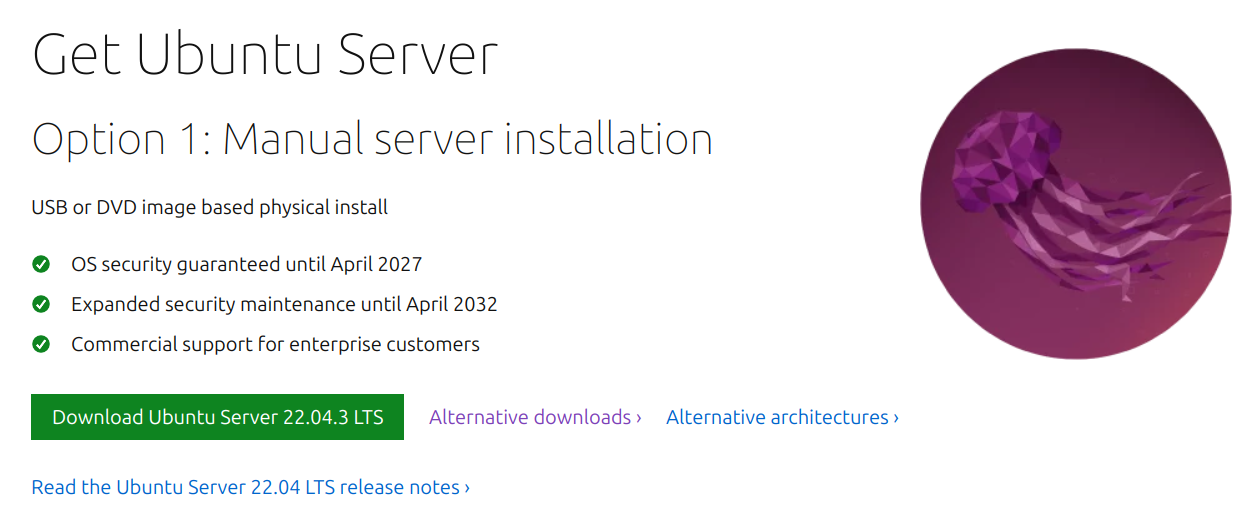


Рис 14. Выбранный дистрибутив.

Запустим созданную ранее виртуальную машину

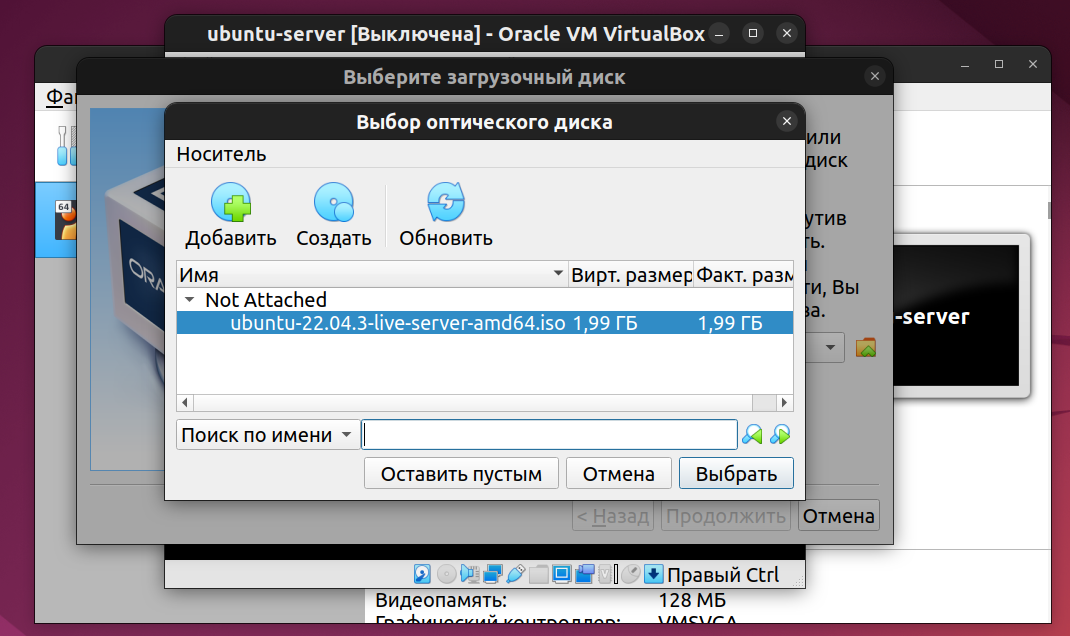


Рис 15. Выбираем ранее установленный iso-образ.

В появившемся Grub-загрузчике выберем Try or install Ubuntu и настроим базовую конфигурацию системы:

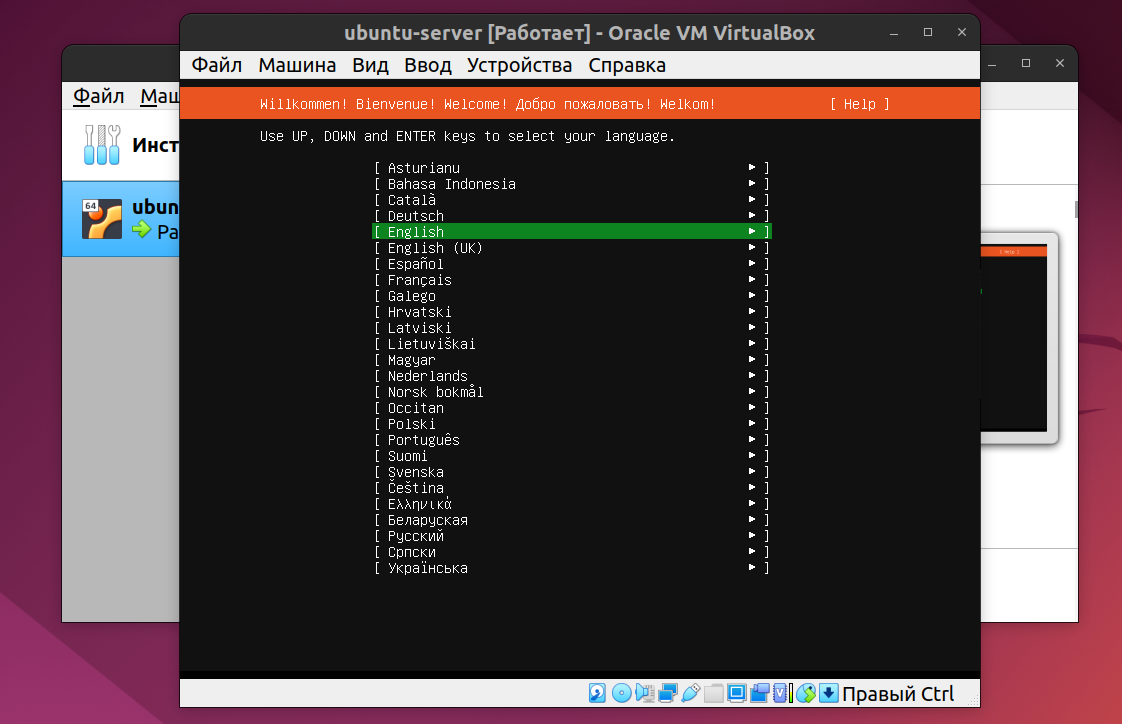


Рис 16. Выбираем английский язык (по умолчанию).

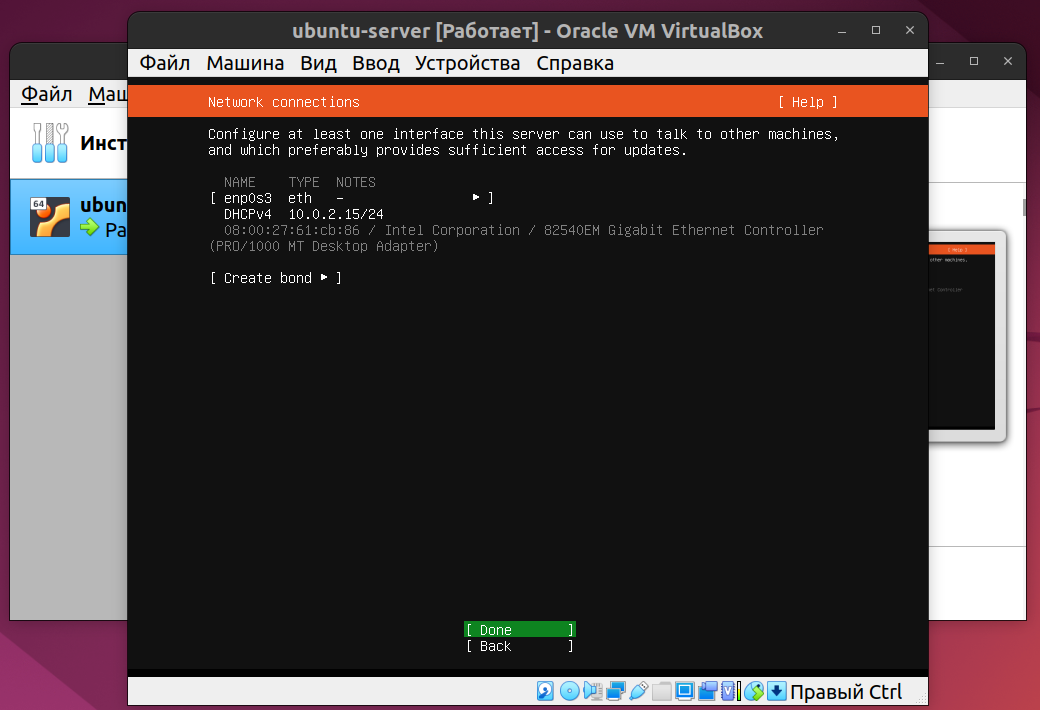


Рис 17. Выбираем сетевое подключение по умолчанию.

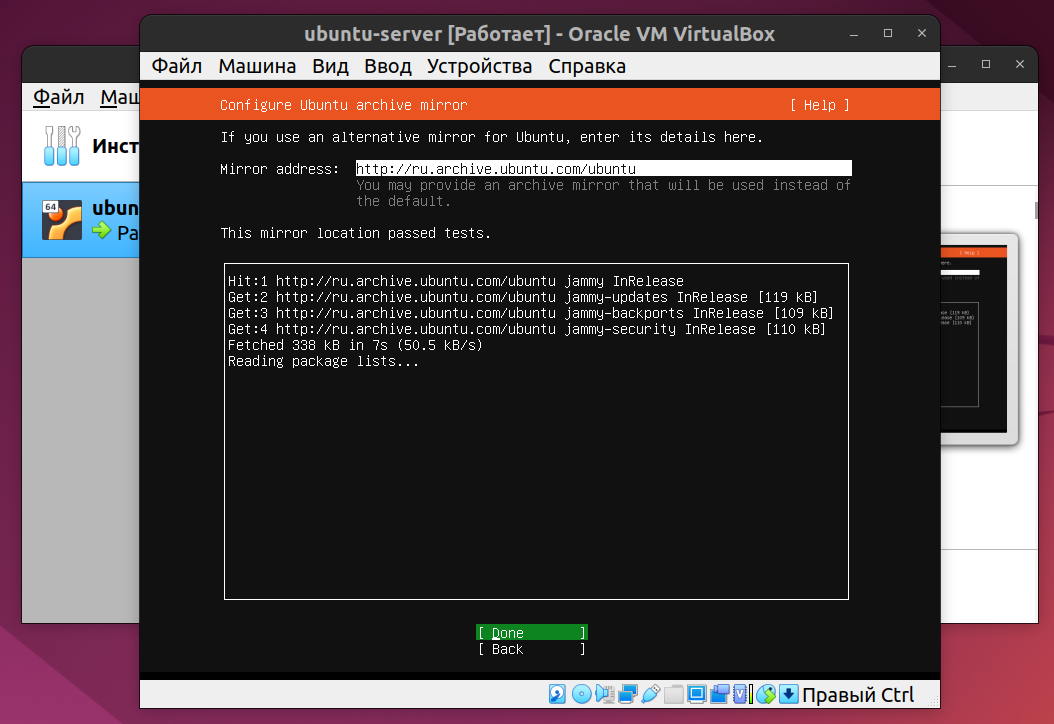


Рис 18. Выбираем с установочными пакетами Ubuntu по умолчанию.

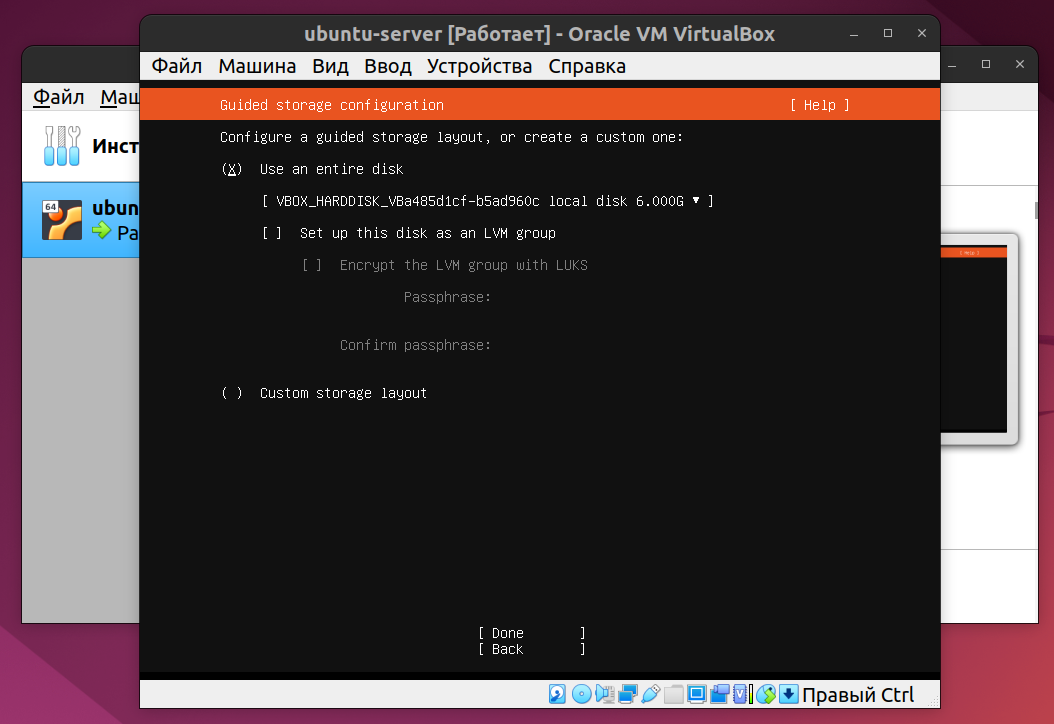


Рис 19. Предоставим установщику весь диск по умолчанию.

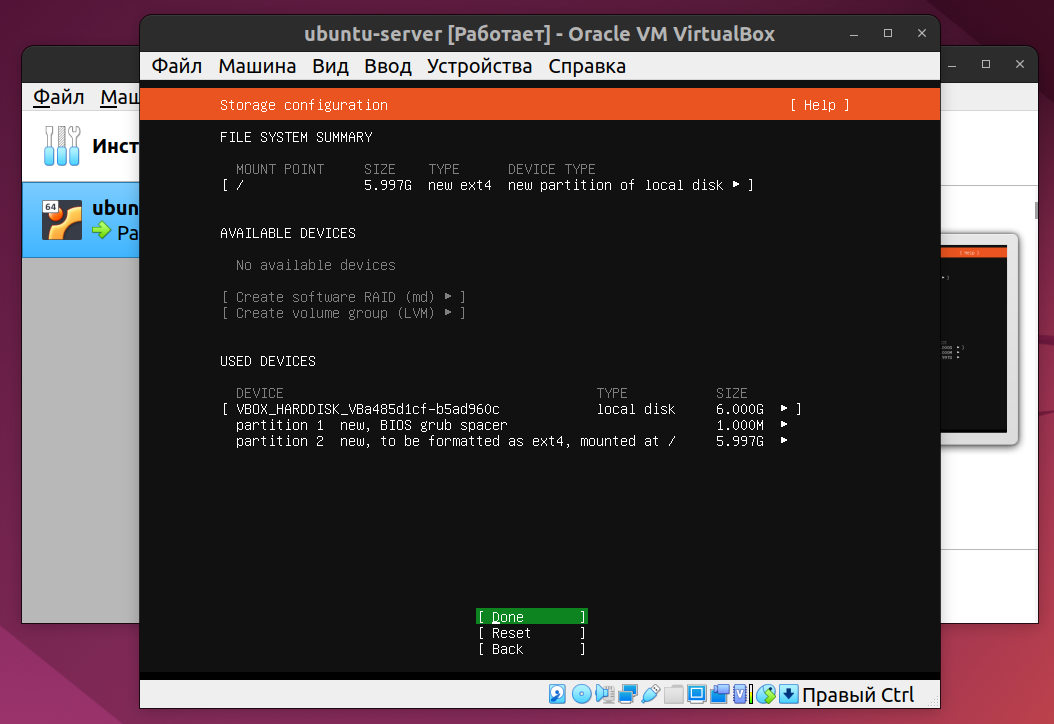


Рис 20. Подтверждаем выбранные настройки.

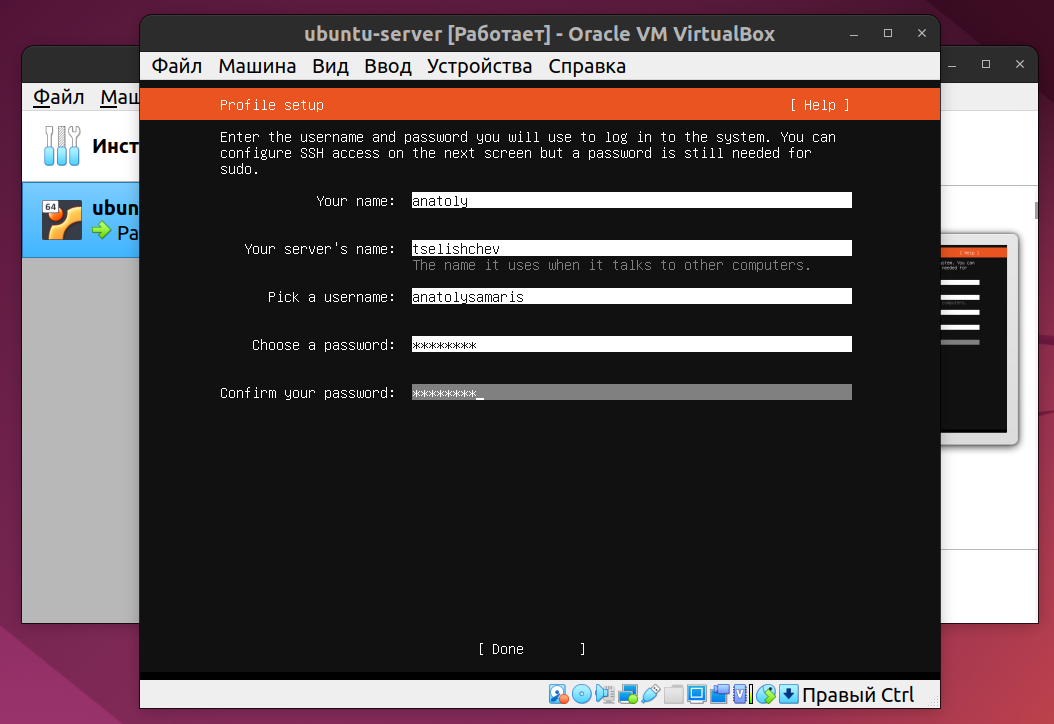


Рис 21. Создаем пользователя.

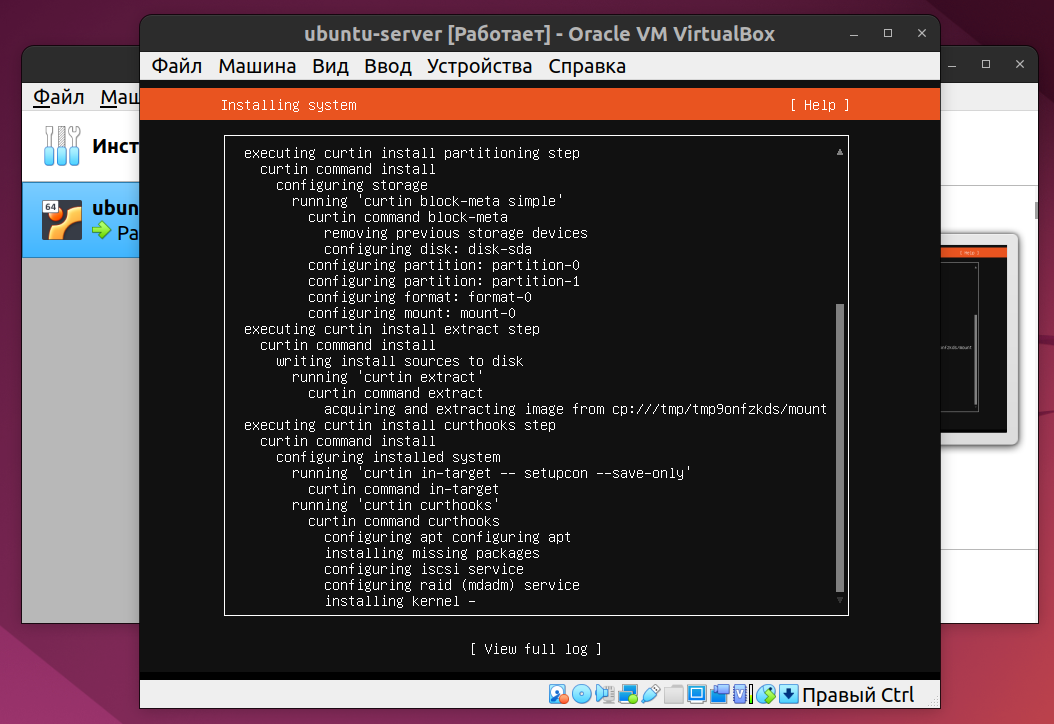


Рис 22. Ждем завершения установки системы.

По завершении установки системы перезагрузим виртуальную машину. После запуска системы введем имя пользователя (в нашем случае anatolysamaris) и пароль пользователя:

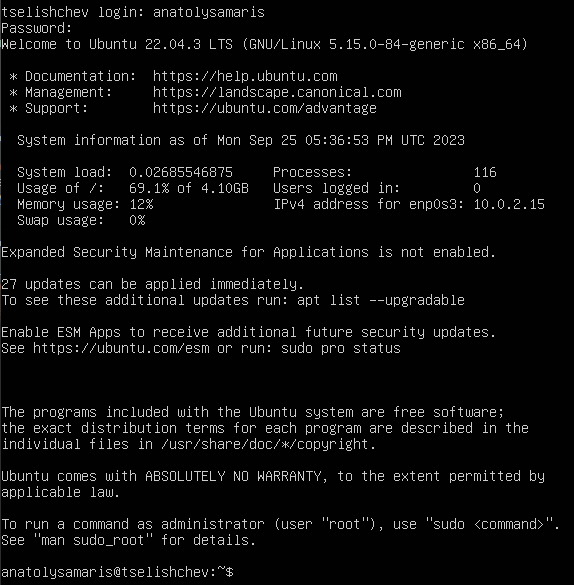


Рис 23. Результат входа в учетную запись.

**Настройка текстового редактора Vim, разработка на Linux**

Так как при установке мы не выбирали минимальную установку, в системе имеются некоторые утилиты и программы, в том числе Vim.

Создадим (в данном случае такой файл не имелся в системе изначально) файл .vimrc в котором опишем некоторые настройки редактора Vim, а также установим несколько плагинов, используя менеджер плагинов vim-plug:

vim .vimrc

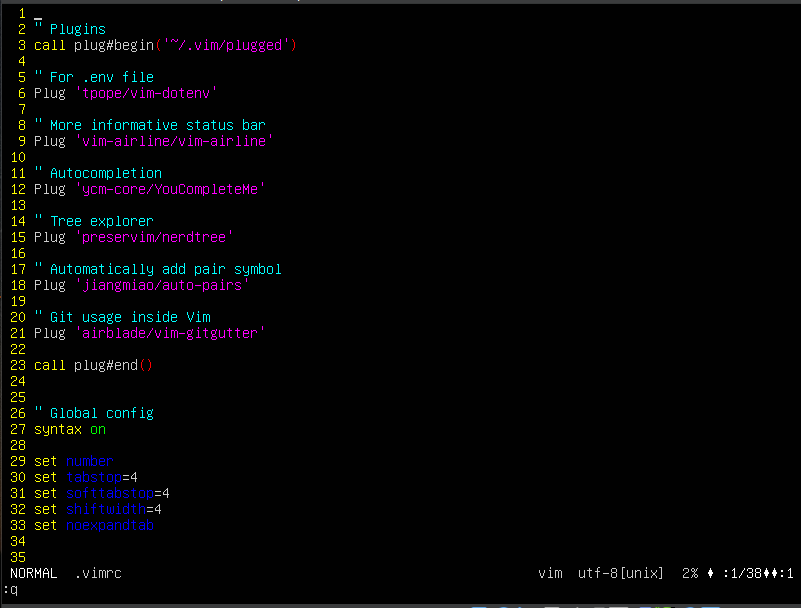


Рис 24. Содержимое файла .vimrc.

Сохраним содержимое, выйдя в командный режим и написав :w, затем введём команду :PlugInstall для установки описанных в файле плагинов:

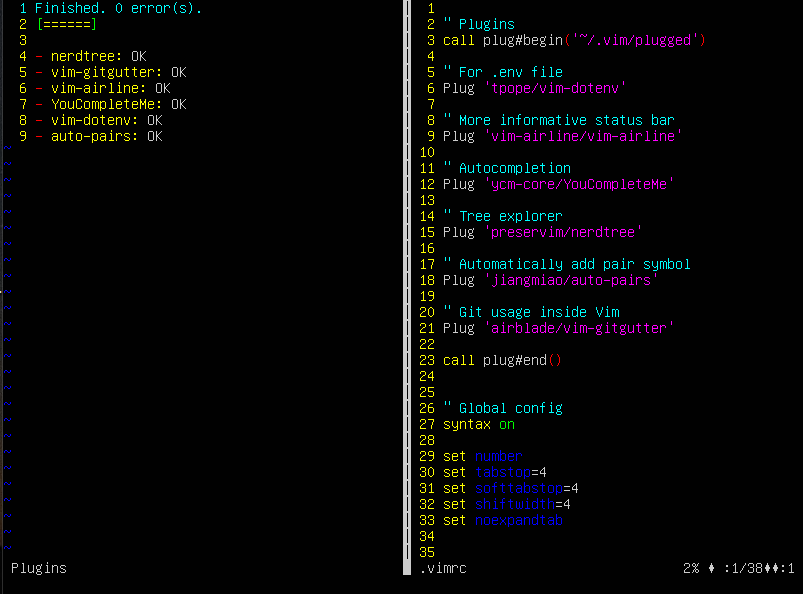


Рис 25. Установка плагинов прошла успешно.

Выйдем из текущего файла, написав :q

Напишем в терминале

vim code.cpp

Будет создан файл с расширением .cpp, в котором напишем код, выполняющий задание по варианту, на языке С++:

|  |  |
| --- | --- |
| Вариант | Задание |
| 6 | Дан одномерный массив из N целочисленных элементов (тип int). Задать произвольное число k в интервале 2..N-2. Удалить элемент с этим номером и сдвинуть элементы, находящиеся после него, на эту позицию. |

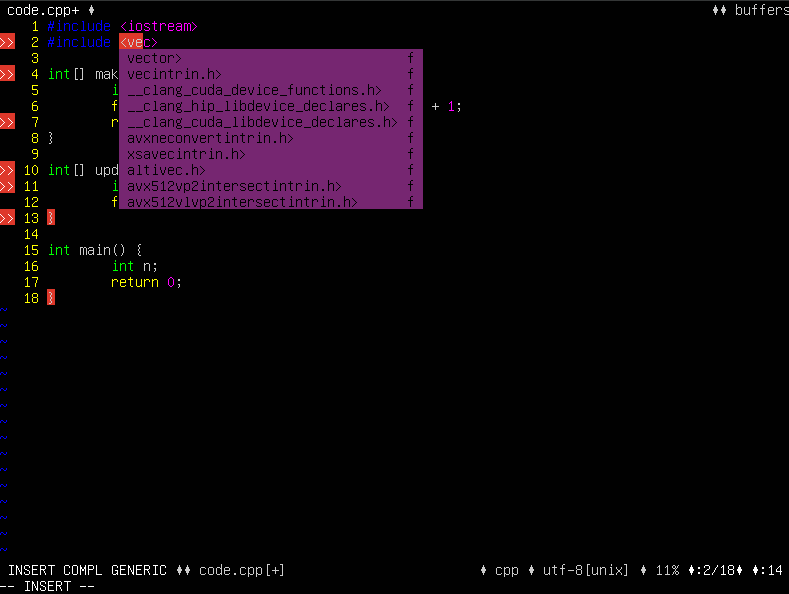


Рис 26. Демонстрация работы автодополнения - одного из установленных плагинов.

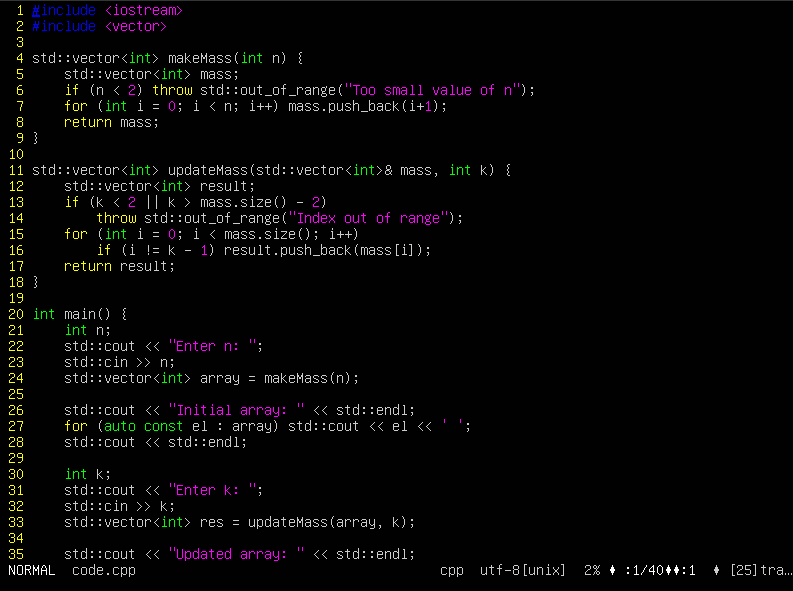


Рис 27. Итоговый код решения.

Скомпилируем программу, используя следующую команду:

sudo g++ -o code code.cpp

В текущей директории появится исполняемый файл code, который можно запустить, введя в терминале:

./code

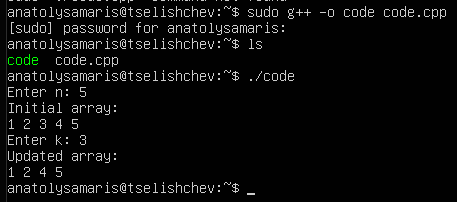


Рис 28. Компиляция и запуск программы.

Для удобства переместим рабочие файлы в новый каталог testProject:

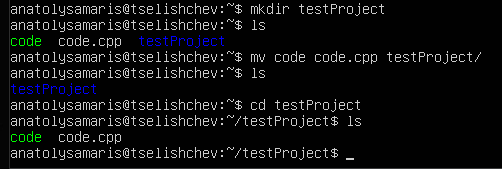


Рис 29. Перемещение файлов в отдельный каталог.

Создадим в этом же каталоге файл makefile со следующим содержимым:

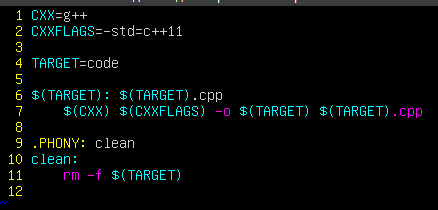


Рис 30. Содержимое файла makefile.

Скомпилируем и запустим код программы, используя утилиту make. Если бы решение располагалось в нескольких файлах, использование этой утилиты сильно упростило бы компиляцию и запуск программы.

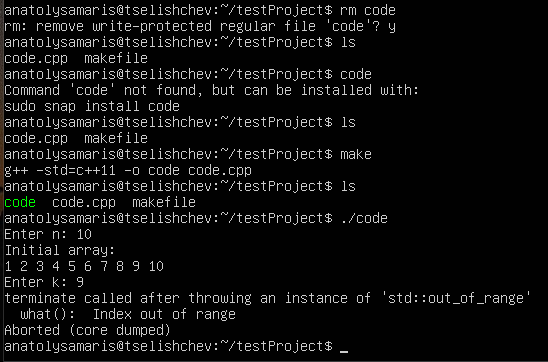


Рис 31. Компиляция и запуск программы с помощью утилиты make.

Таким образом, всё работает корректно при обоих способах компиляции.

**Выводы**

- Установлен гипервизор Oracle VM VirtualBox;

- Создана и настроена виртуальная машина;

- Установлена серверная ОС Ubuntu LTS 22.04;

- Настроен текстовый редактор Vim, установлены плагины с использованием менеджера плагинов vim-plug;

- Написана и скомпилирована программа на С++, выполняющая задание по варианту, с использованием компилятора g++ и с помощью утилиты make.

**Ответы на контрольные вопросы**

**Что такое IDE?**

IDE это программное обеспечение, объединяющее в едином интерфейсе инструменты разработки и отладки приложений.

**Что такое API?**

Интерфейс программирования приложений: набор готовых процедур, функций, структур, с помощью которых разработчик может взаимодействовать с сервисом в собственных приложениях.

**Что такое библиотека в программировании?**

Это готовый набор функций, классов, типов. Обычно они объединены некоторым общим назначением (например, библиотека для работы с графикой).

**Понятия статической и динамической библиотеки.**

Статическая библиотека - файл с программным кодом, описывающим необходимые функции, классы, который подключается к программе с помощью специальной команды. При запуске программы код библиотеки “вставляется” в программу и становится ее частью.

Динамическая библиотека - файл с кодом, который загружается в программу во время ее выполнения. То есть при компиляции программы динамическая библиотека не становится частью программы, она остается отдельной единицей.

**Что такое плагин?**

Плагин - модуль, добавляющий в некоторый функционал в существующую программу. Эти модули независимы и подключаются динамически во время исполнения основной программы.

**Назовите несколько консольных текстовых редакторов для Linux**

Vim, Nano, Emacs

**Что делает команда gcc?**

С помощью данной команды происходит компиляция кода на языке С с помощью компилятора GCC. Ее синтаксис имеет следующий вид:

gcc [параметры] имена файлов

В то же время для компиляции кода на языке С++ используется похожая команда g++.

**Что делает команда make?**

Считывает инструкции из файла Makefile в выбранной директории и выполняет команды, необходимые для компиляции и сборки программы. На самом деле make - это утилита для автоматизации сборки программ из исходных файлов. В файлах Makefile описываются зависимости между файлами и команды для обновления каждого файла.

**Дайте определение заголовочного файла и файла реализации**

Заголовочный файл содержит объявления классов, их полей и методов; функций, переменных и иных структур данных, при этом не содержит их реализацию. Используется для описания интерфейса программы, то есть конкретного способа взаимодействия пользователя (разработчика) с программой. Так, например, тестировщик, зная интерфейс функции Х, будет иметь возможность написать для неё юнит-тесты, даже если описанная функция еще не реализована.

Файл реализации содержит описание функций, классов и их методов, и иных структур данных, описанных в соответствующем заголовочном файле. Другими словами, здесь содержится реализация описанного в заголовочном файле интерфейса. Пример:

*example.h*

// Объявление функции, считающей значение дискриминанта по трем параметрам

double countDiscriminant(double a, double b, double c);

*example.cpp*

#include “example.h”

// Реализация (описание) объявленной выше функции

double countDiscriminant(double a, double b, double c) {

discriminant = b \* b - 4 \* a \* c;

return discriminant;

}

**Что означает единица трансляции? В чем особенность разработки программ из нескольких единиц трансляции?**

В единицу трансляции включены файл реализации (.с/.срр) и все его заголовочные файлы (.h/.hpp). По сути это отдельный файл/модуль программы, который может быть скомпилирован отдельно от других модулей. Например, два файла, описанные в ответе на предыдущий вопрос, составляли бы единицу трансляции и препроцессор передал бы компилятору следующий код:

// Объявление функции, считающей значение дискриминанта по трем параметрам

double countDiscriminant(double a, double b, double c);

// Реализация (описание) объявленной выше функции

double countDiscriminant(double a, double b, double c) {

discriminant = b \* b - 4 \* a \* c;

return discriminant;

}

Особенность разработки программ из нескольких единиц трансляции заключается в том, что это позволяет разделить программу на логические модули, каждый из которых может быть разработан и отлажен независимо от других, что упрощает процесс разработки и поддержки ПО, а также повышает переиспользуемость кода.

**Дайте краткую характеристику каждому этапу трансляции программ, написанных на Си**

1. Текстовый препроцессор

Обрабатывает исходный код программы, выполняя директивы препроцессора, такие как #include, #define и тд.

1. Компилятор

Преобразует исходный код программы в объектный код, который представляет собой машинный код, но ещё не зависит от конкретной платформы.

1. Линковщик

Объединяет объектные файлы и библиотеки в один исполняемый файл, разрешая ссылки на функции и переменные, которые определены в других файлах.