Министерство образования и науки Челябинской области

государственное бюджетное профессиональное

образовательное учреждение

«Златоустовский индустриальный колледж им. П.П. Аносова»

**ЗАЩИТА**

Руководитель УП ПМ.03

Преподаватель ГБОУ ЗлатИК

\_\_\_\_\_\_\_\_\_\_\_\_\_\_Ю.В.Майер

Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата\_\_\_\_\_\_\_\_\_\_\_\_

**ОТЧЕТ**

По учебной практике

Специальность: 09.02.07 «Информационные системы и программирование»

ПМ.03 «Ревьюирирование программных продуктов»

Выполнил:

Студент группы\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_В.Е.Левин

\_\_\_\_\_\_\_\_\_\_\_дата

2021-2022 уч.г.

Содержание

[Введение 3](#_Toc88812348)

[1. Список терминов и определений 5](#_Toc88812349)

[2. Функциональные требования 6](#_Toc88812350)

[3. Измерение производительности приложения посредством анализа использования ЦП 7](#_Toc88812351)

[4. Анализ данных о загрузке ЦП 11](#_Toc88812352)

[5. Просмотр внешнего кода 13](#_Toc88812353)

[6. Установка программного обеспечения 15](#_Toc88812354)

[7. Дизассемблирование 1](#_Toc88812357)7

[8. Обратное проектирование 22](#_Toc88812358)

[ЗАКЛЮЧЕНИЕ 24](#_Toc88812359)

[ЛИТЕРАТУРА 25](#_Toc88812364)

[ПРИЛОЖЕНИЕ 1 2](#_Toc88812360)6

[ПРИЛОЖЕНИЕ 2 2](#_Toc88812361)7

[ПРИЛОЖЕНИЕ 3 30](#_Toc88812362)

[ПРИЛОЖЕНИЕ 4 31](#_Toc88812363)

# Введение

Процессы разработки, приобретения и внедрения сложных систем, к которым относятся в частности программные комплексы, должны находится под жестким управленческим контролем. В настоящее время практически во всех организациях обеспечивается контроль важнейших характеристик, связанных с производством и использованием программных продуктов, таких как время, финансовые средства, ресурсы и т.п. Однако в большинстве случаев вне пределов сферы контроля оказывается наиболее важная характеристика программных продуктов, ради которой, собственно и осуществляются затраты времени, финансовых средств и ресурсов – это качество продукта, поскольку «невозможно контролировать то, что нельзя измерить» (“You cannot control what you cannot measure”).

Дизассемблирование **–** преобразованиепрограммы на машинном языке к ее ассемблерному представлению. Декомпиляция – получение кода языка высокого уровня из программы на машинном языке или ассемблере.

Под *анализом потоков данных* понимают совокупность задач, нацеленных на выяснение некоторых глобальных свойств программы, то есть извлечение информации о поведении тех или иных конструкций в некотором контексте.

Основным результатом деятельности группы разработчиков являются не диаграммы, а программное обеспечение, поэтому модели и основанные на них реализации должны соответствовать друг другу с минимальными затратами по поддержанию синхронизации между ними. Чаще всего разработанные модели преобразуются в программный код. Хотя UML не определяет конкретного способа отображения на какой-либо объектно-ориентированный язык, он проектировался с учетом этого требования. В наибольшей степени это относится к диаграммам классов, содержание которых без труда отображается на такие известные объектно-ориентированные языки программирования, как Java, C++, ObjectPascal, Visual Basic и др.

*Прямым проектированием*(Forward engineering) называется процесс преобразования модели в код путем отображения на некоторый язык реализации.

*Обратным проектированием*(Reverse engineering) называется процесс преобразования в модель кода, записанного на каком-либо языке программирования.

Цель учебной практики: Научиться выполнять прямое и обратное проектирование программного обеспечениия, овладеть навыками приостановки и возобновления работы с рабочими задачами иинспекцией кода в Visual Studio, совместно работать над проектом в системе контроля версий GIT.

В ходе учебной практики для достижения цели, ставятся задачи:

* измерить производительность приложения посредством анализа использования ЦП,
* установить программного обеспечение (Linux, компилятор GCC, radare2, iaito) для обратного проектирования,
* выполнить обратное проектирование,
* выполнить дизассемблирование.

# Список терминов и определений

*Точка останова* - это преднамеренное прерывание выполнения программы, при котором выполняется вызов отладчика (одновременно с этим программа сама может использовать точки останова для своих нужд). После перехода к отладчику программист может исследовать состояние программы (логи, состояние памяти, регистров процессора, стека и т. п.), с тем чтобы определить, правильно ли ведёт себя программа. В отличие от полной остановки, с помощью останова, после работы в отладчике программа может быть завершена либо продолжена с того же места, где произошёл останов.

*Репозиторий* (от англ. Repository - хранилище) — место, где хранятся и поддерживаются какие-либо данные. Чаще всего данные в репозитории хранятся в виде файлов, доступных для дальнейшего распространения по сети.

GitHub - это система управления проектами и версиями кода, а также платформа социальных сетей, созданная для разработчиков.

*Radare2* (Также известный как r2) – свободный кроссплатформенный фреймворк для реверс-инжиниринга, написанный на Си, который включает дизассемблер, шестнадцатеричный редактор, анализатор кода и т.п. Используется при реверсе, отладке вредоносного ПО и прошивок

*Декомпилятор* — это программа, транслирующая [исполняемый модуль](https://ru.wikipedia.org/w/index.php?title=%D0%98%D1%81%D0%BF%D0%BE%D0%BB%D0%BD%D1%8F%D0%B5%D0%BC%D1%8B%D0%B9_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8C&action=edit&redlink=1) (полученный на выходе [компилятора](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80)) в эквивалентный [исходный код](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) на [языке программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) высокого уровня.

*Декомпиляция* – получение кода языка высокого уровня из программы на машинном языке или ассемблере.

*Обратное проектирование (Reverse Engineering)* - исследование некоторого готового устройства или программы, а также документации на него с целью понять принцип его работы; например, чтобы обнаружить недокументированные возможности (в том числе программные закладки), сделать изменение или воспроизвести устройство, программу или иной объект

*Дизассемблирование* **–** преобразованиепрограммы на машинном языке к ее ассемблерному представлению.

*Компилятор* — программа, переводящая текст, написанный на языке программирования, в набор машинных кодов

# Функциональные требования

Для выполнения поставленных задач, мы использовали программное обеспечение Linux в нём мы устанавливали пакеты для работы с radare2, так же устанавливали графический интерфейс и приложение «Iaito».

Для выполнения задачи с измерение производительности приложений, нам нужна была программа Visual Studio [(см. в приложение 1).](#_ПРИЛОЖЕНИЕ_1_2)

Для сохранения наших проектов мы использовали систему контроля версий GitHub [(см. приложение 2).](#_ПРИЛОЖЕНИЕ_2_1)

# Измерение производительности приложения посредством анализа использования ЦП

1. Запустили Visual Studio для будущего проекта, в котором работаем

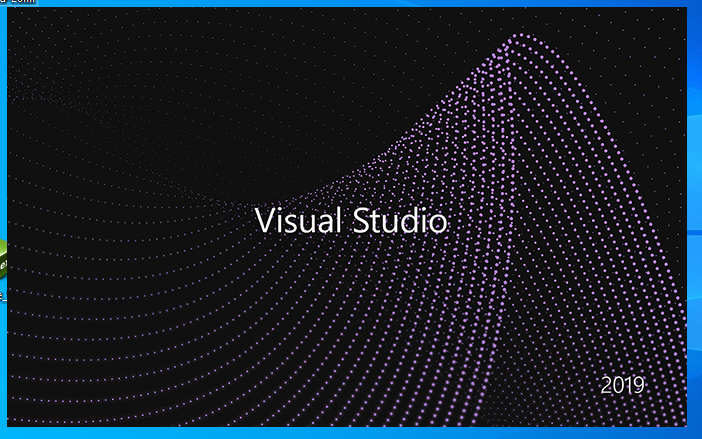


Рис. 1 Окно запуска Visual Studio

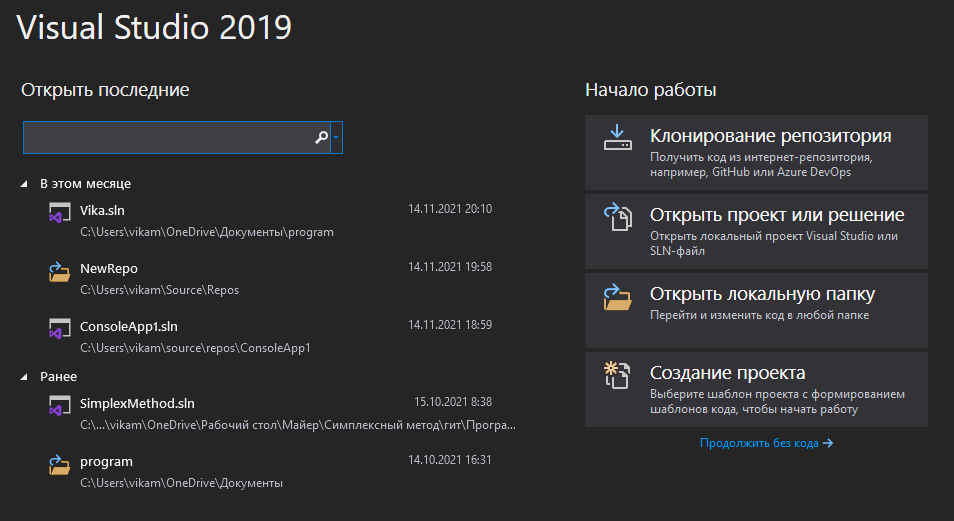


Рис. 2 Окно выбора в проекте

1. Открываем проект, в котором будем работать с ЦП

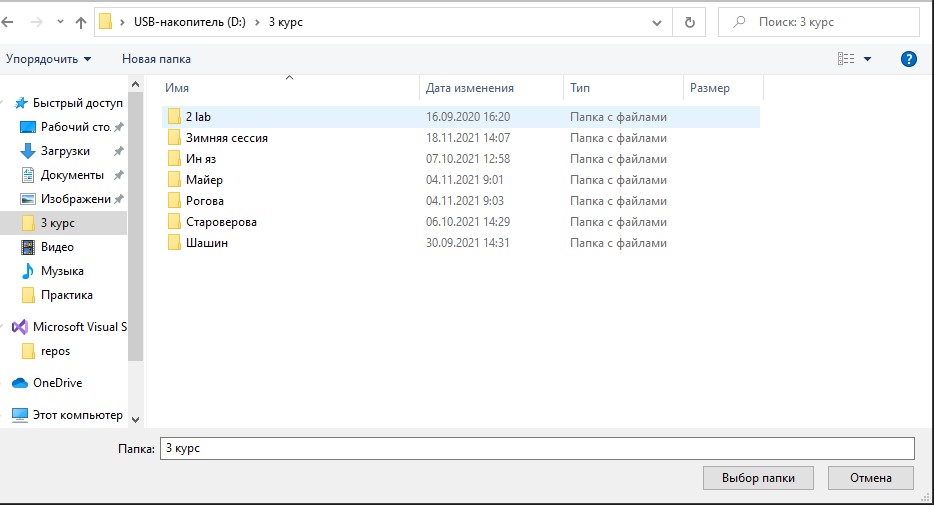


Рис. 3 Выбор папки для сохранения проекта

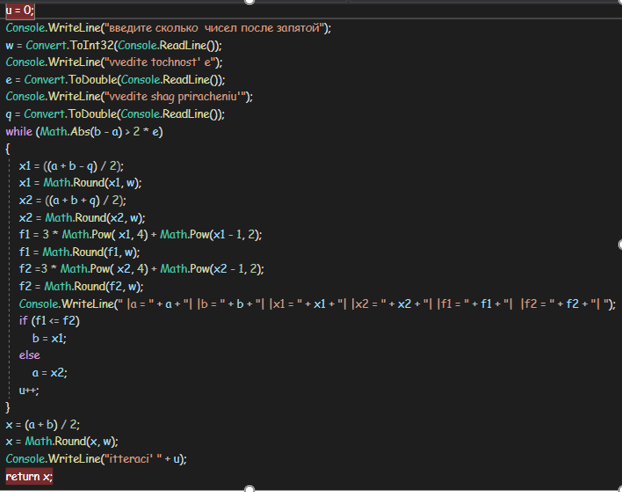


Рис. 4 наш проект

1. Открыли проект и установили точки останова, в любой точке проекта, либо же в начале функции.

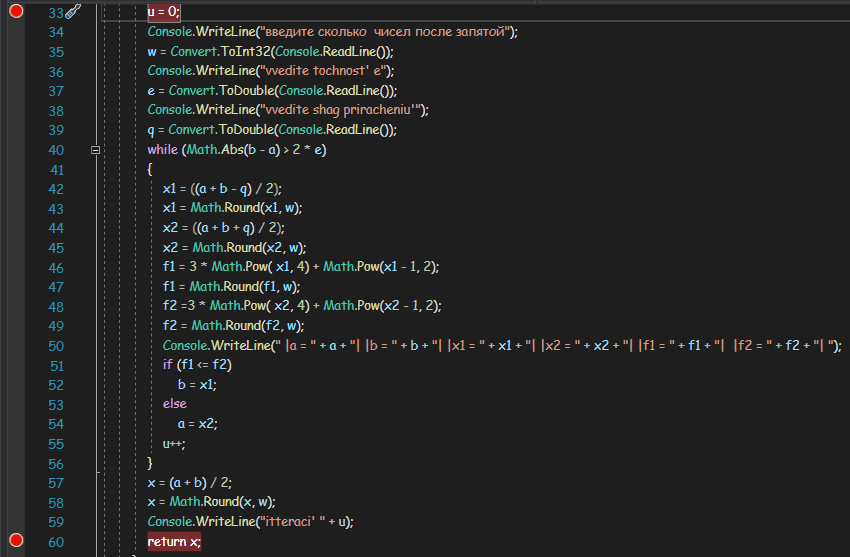


Рис. 5 Точки останова в проекте

1. Ставим вторую точку останова в конце функции



Рис. 6 Вторая точка останова

1. Открыли *Отладка>Окна>Показать средства диагностики*

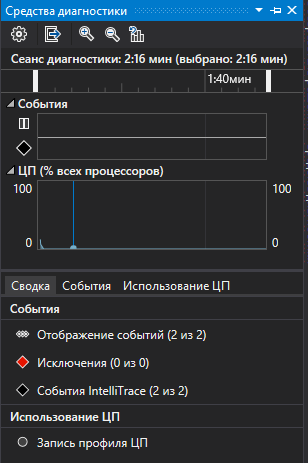


Рис. 7 Средства диагностики ЦП

1. Вы можете выбрать, что следует просмотреть, [Использование памяти](https://docs.microsoft.com/ru-ru/visualstudio/profiling/memory-usage?view=vs-2022) или **Загрузка ЦП** (либо оба средства), с помощью параметра **Выбор средств** на панели инструментов. В Visual Studio Enterprise также можно включить или отключить IntelliTrace, выбрав ***Сервис****>****Параметры****>****IntelliTrace****.*

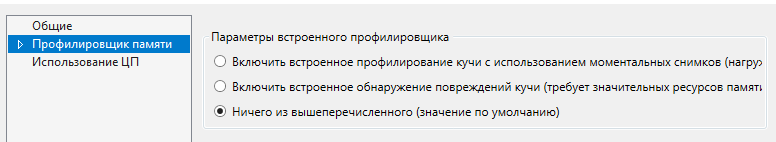


Рис. 8 IntelliTrace

1. Щелкните ***Отладка > Начать отладку***(**Запустить** на панели инструментов или **F5**).



Рис. Отладка программы

1. Запустите сценарий, который вызвал срабатывание первой точки останова.



Рис. 10 Срабатывание первой точки останова

1. Приостановив отладчик, включите сбор данных о загрузке ЦП, а затем откройте вкладку **Загрузка ЦП**.

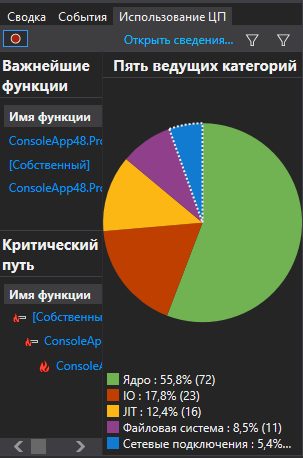


Рис. 11 Использование ЦП

1. Нажмите клавишу F5, чтобы запустить приложение до второй точки останова.

Теперь у вас есть данные о производительности приложения именно для той области кода, которая выполняется между двумя точками останова.

Профилировщик начинает подготавливать данные потока. Дождитесь завершения этой операции.

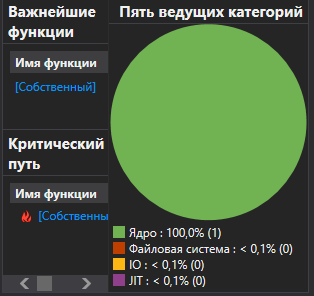


Рис. 12 Нагрузка второй точки останова

Если мы хотим выбрать для анализа более конкретную область кода, выберите область на временной шкале ЦП (это должна быть область, в которой отображаются данные профилирования).

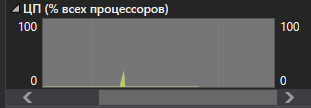


Рис. 13 Выбор области на шкале ЦП

### 4. Анализ данных о загрузке ЦП

1. В списке функций изучите функции, которые выполняют большую часть работы.

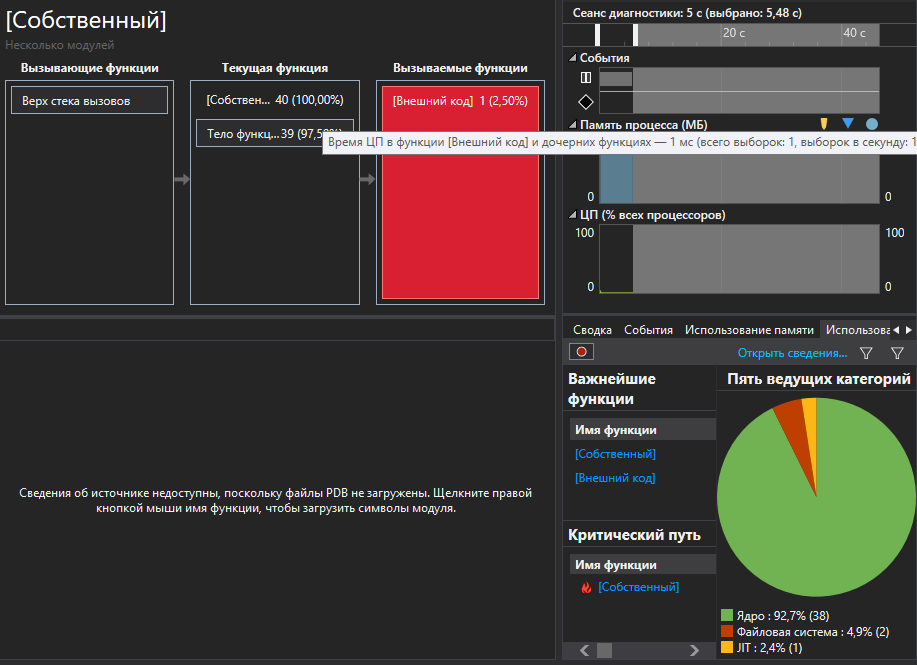


Рис. 14 Список функций, выполняющая большую часть работы

В списке функций дважды щелкните одну из функций вашего приложения, которая выполняет много работы.

При двойном щелчке функции в левой панели откроется представление **Вызывающий/вызываемый.**

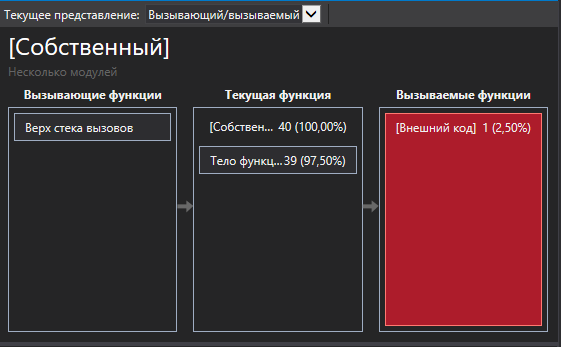


Рис. 15 Вызывающая/вызываемая функция

В этом представлении функции отображается в заголовке и в поле Текущая функция. Функция, вызывавшая текущую функцию, отображается в левой части окна в разделе Вызывающие функции, а все функции, вызываемые текущей функцией, отображаются в поле Вызываемые функции справа.

В этом представлении показано общее время (мс) и доля общего времени выполнения приложения, затраченного на выполнение функции. В поле Тело функции также показан общий объем времени (и доля времени), затраченного в теле функции за исключением времени, затраченного в вызываемых и вызывающих функциях. (В этом примере в теле функции затрачено 39 из 40 мс, а оставшиеся 1 мс затрачены во внешнем коде, вызванном этой функцией).

2. Чтобы увидеть более обобщенное представление, показывающее порядок, в котором вызываются функции, выберите в раскрывающемся списке в верхней части панели пункт Дерево вызовов.

Каждая нумерованная область на рисунке соответствует определенному шагу в процедуре.

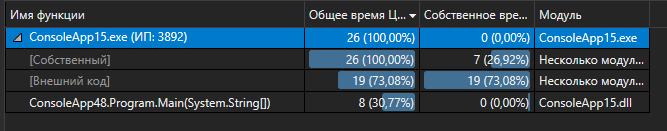
****

Рис. 16 Шаг в процедуре

Чтобы увидеть вызовы функций, которые используют самый высокий процент ЦП в представлении дерева вызовов, нажмите **«развернуть критический путь».**

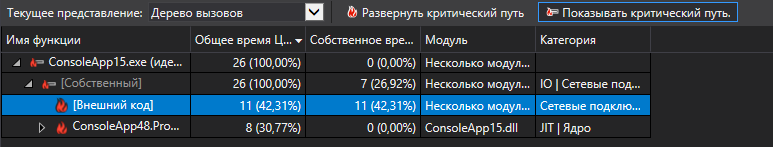


Рис. 17 Развернутый критический путь

**5. Просмотр внешнего кода**

Внешний код — это функции в компонентах системы и платформы, которые исполняются вашим кодом. Внешний код включает функции, которые запускают и останавливают приложение, отрисовывают пользовательский интерфейс, управляют потоками и предоставляют приложению другие низкоуровневые службы. В большинстве случаев внешний код вас интересовать не будет, поэтому средство "Загрузка ЦП" собирает внешние функции пользовательского метода в один узел [Внешний код] .

Если вы захотите посмотреть пути к вызовам внешнего кода, выберите Показать внешний код в списке Представление фильтра и выберите Применить.

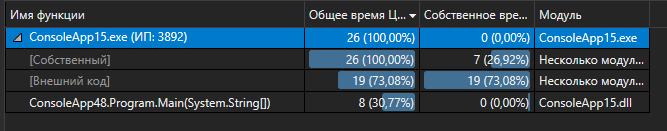
****

Рис. 18 Показ внешнего кода

Работу смотреть в GitHub ([см. приложение 3](#_ПРИЛОЖЕНИЕ_3))

# 6. Установка программного обеспечения

### **6. 1. Установка программного обеспечения VirtualBox**

Установка показана в [приложении 4](#_ПРИЛОЖЕНИЕ_4)

### 6. 2. Создание исходного файла

1. Устанавливаем компилятор GCC и все необходимые для него компоненты

sudo apt install build-essential manpages-dev git automake autoconf

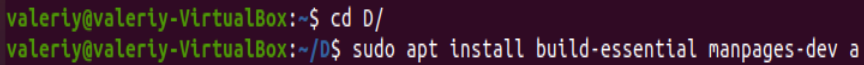


Рис. 19 Установка GCC

1. Устанавливаем необходимые пакеты для работы с radare2

sudo apt install git build-essential make

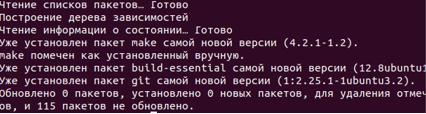


Рис. 20 Пакеты для работы с Radare2

1. Установка radare2:

git clone https://github.com/radareorg/radare2

radare2/sys/install.sh

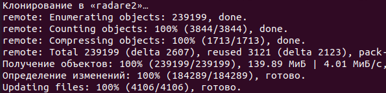


Рис. 21 Установка Radare2

1. Проверяем и обновляем radare2

r2pm update



Рис. 22 Проверка и обновление Radare2

1. Устанавливаем необходимые компоненты для граф.оболочки «Iaito»

sudo apt install qttools5-dev-tools qtbase5-dev qtchooser qt5-qmake qtbase5-dev-tools libqt5svg5-dev make

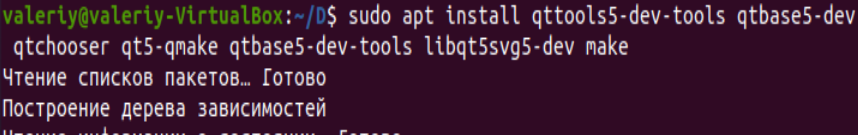


Рис. 23 Компоненты для Iaito

Устанавливаем графический интерфейс «Iaito» (установка может выполняться из любого места)

r2pm install iaito



Рис. 24 Установка Iaito

1. Устанавливаем декомпилятор «r2ghidra» (установка может выполняться из любого места)

r2pm install r2ghidra



Рис. 25 Установка R2Ghidra

# 7. Дизассемблирование

Для выполнения задания с дизассемблирование мы использовали установленное приложение Iaito.

Мы писали программу, которая выводит 4 раза слово «Дизассемблирование», в ходе работы нам надо исправить ошибку с помощью приложения Iaito дизассемблирование, что бы программа выводила 10 раз.

1. Запускаем «Iaito»

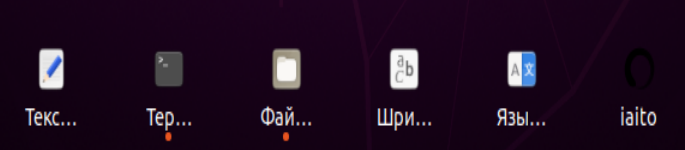


Рис. 26 Запуск Iaito

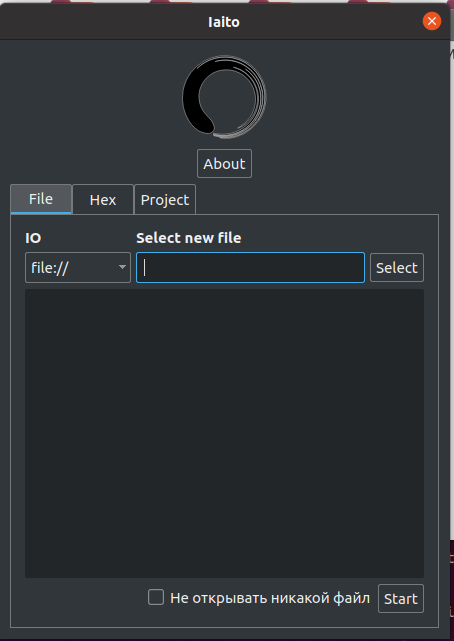


Рис. 27 Главное окно Iaito

1. Запускаем и проверяем тестовый файл (после можно закрыть)

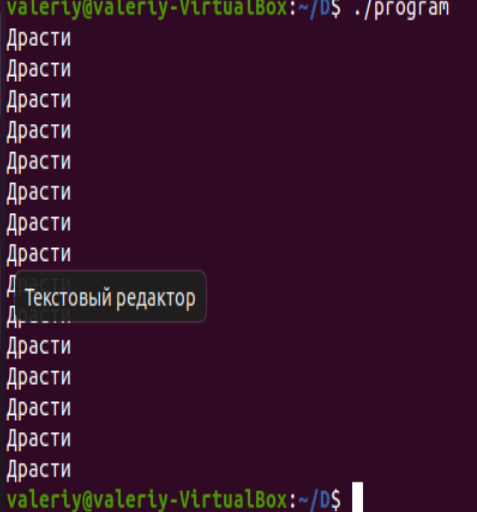


Рис. 28 Проверка компиляции

1. Открываем тестовый файл в «radare2». Настройки оставляем стандартные

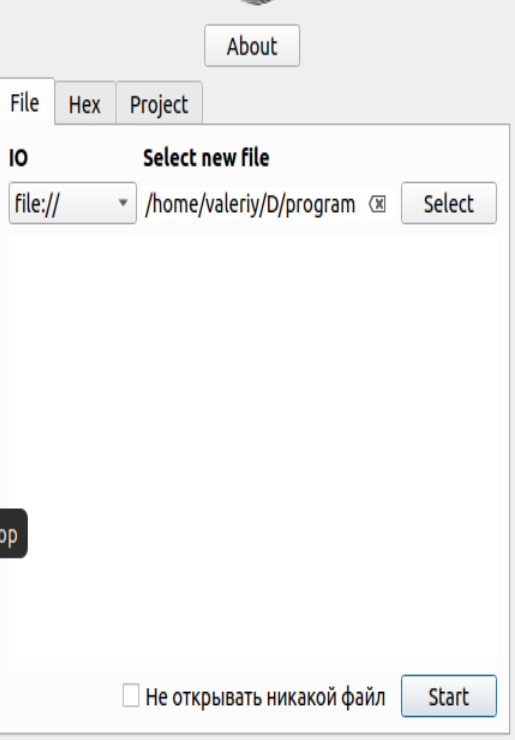


Рис. 29 Открытие текстового файла в Radare2

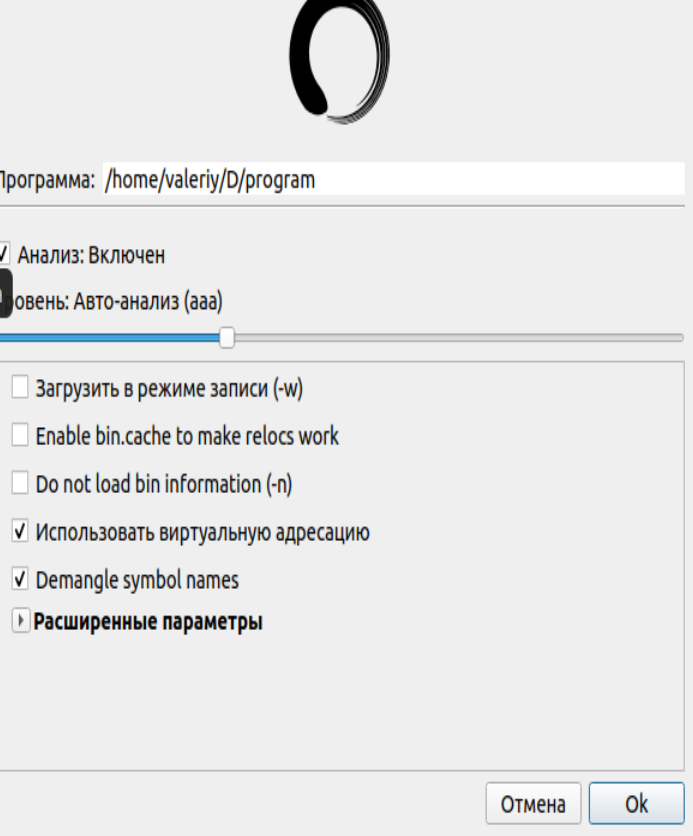


Рис. 30

1. Открываем слева «main»

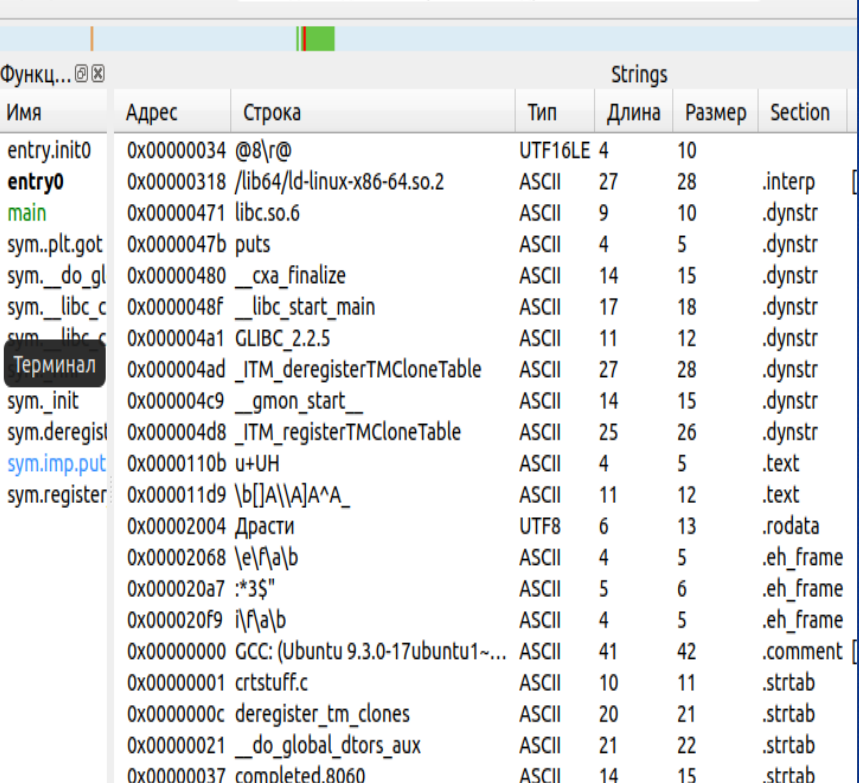


Рис. 31 Открытие Main

1. Открываем вкладку декомпилятора. Видим, что код практически читаем, кроме переменных

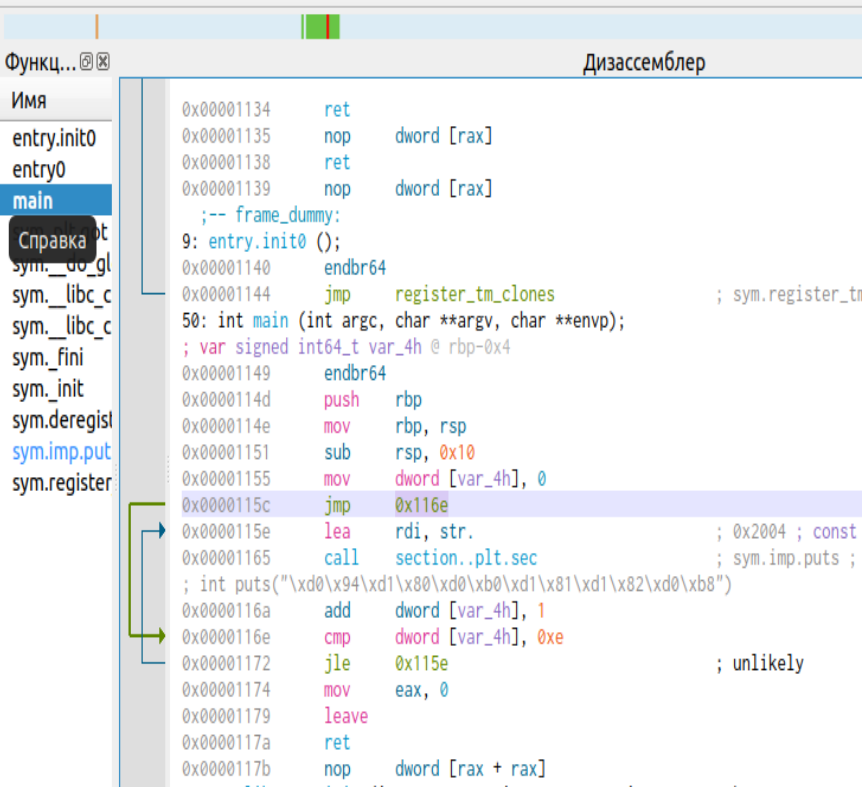


Рис. 32 Вкладка декомпилятора

1. Меняем режим работы программы (File > Set mode > Cache mode)

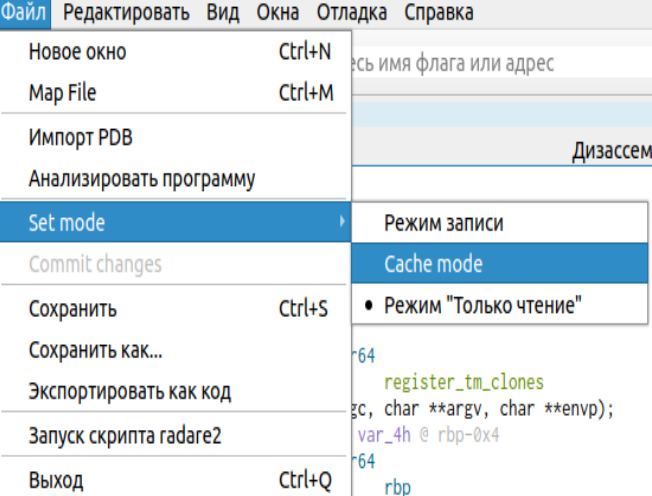


Рис. 33 Меняем режим работы

1. Так как фраза «Драсти» выводится 14 раз, а нам необходимо уменьшить вывод до 4 раз, то смотря на код, можно увидеть, что используется цикл «while» по значение переменной [var15\_h] равной 15 (т.к While < 15, а цикл начинается с 0). Выделяем цифру 14 и нажимаем «Edit» меняя значение переменное с 14 до 4.

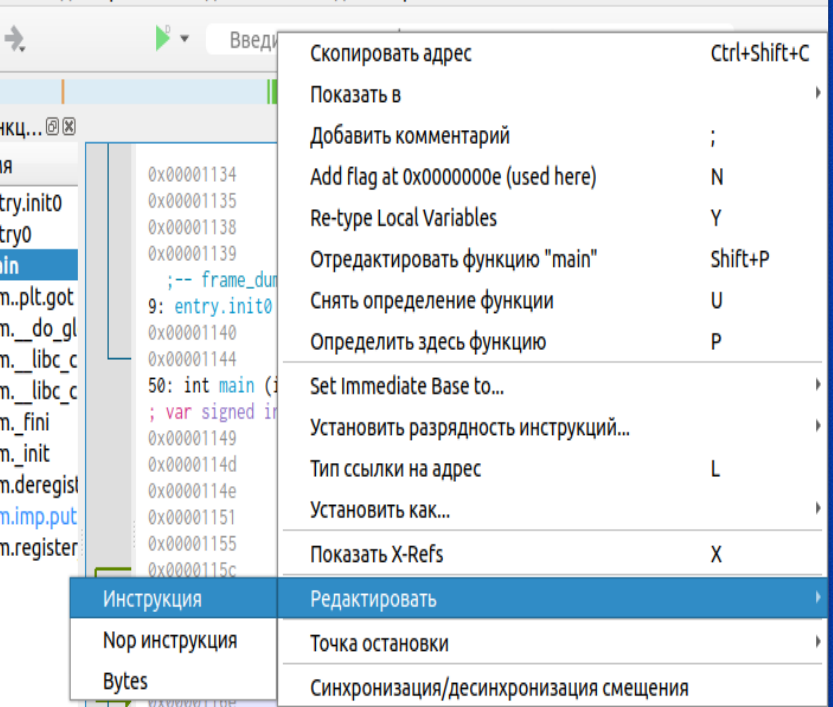


Рис. 34

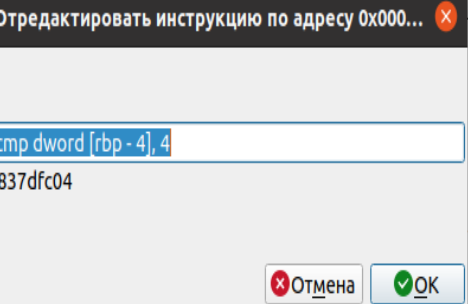


Рис. 35

1. Сохраняем наши изменения, нажав на кнопку «Commit changes»

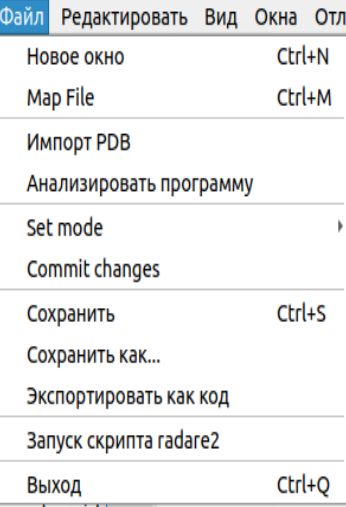


Рис. 36 Сохраняем изменения

1. Запускаем наш файл для проверки

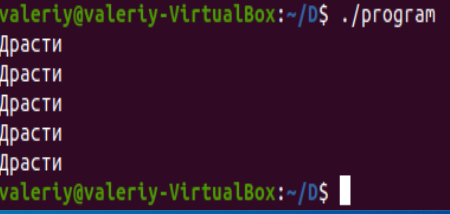


Рис. 37 Компиляция, которую переделали

Работу смотреть в GitHub ([см. приложение 3](#_ПРИЛОЖЕНИЕ_3))

# 8. Обратное проектирование

1. Создаем файл с кодом для компиляции 101

touch program.с



Рис. 38 Создание файла

1. Открываем его и вставляет туда код для теста

[#include](https://vk.com/im?sel=320239488&st=%23include) <stdio.h>  
  
int main (void)  
{  
int x = 0;  
while(x < 15){  
printf ("Драсти\n");

x+=1;  
}  
return 0;  
}

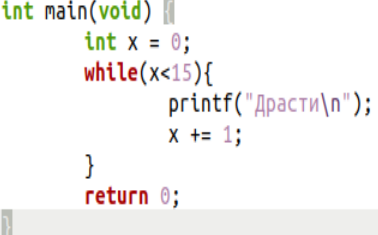


Рис. 39 Код, с которым работаем

1. Компилируем

gcc program.c -o program



Рис. 40 Компиляция

1. Открываем и проверяем наш файл

./program

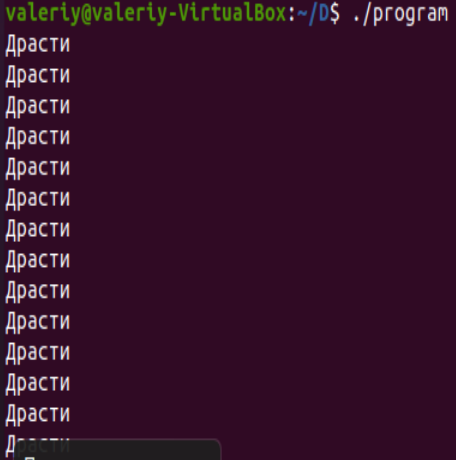


Рис. 41Вывод

1. Открываем папку и открываем файл



Рис. 42 Вход в Radare2



Рис. 43



Рис. 44

1. Запускаем наш файл для проверки

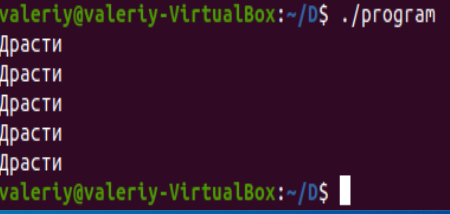


Рис. 45 Проверка файла

# ЗАКЛЮЧЕНИЕ

Целью учебной практики по ПМ.03 «Ревьюирирование программных продуктов» являлось, научиться выполнять прямое и обратное проектирование программного обеспечения, овладеть навыками приостановки и возобновления работы с рабочими задачами иинспекцией кода в Visual Studio, совместно работать над проектом в системе контроля версий GIT.

В ходе выполнения заданий на учебную практику было выполнено изучение терминов для большего понимания работы, таких как репозиторий, дизассемблер, GitHub, radare2, компилятор, GCC, Программное обеспечение, обратная разработка, Linux. Узнали как измерять производительность приложения посредством анализа использование ЦП. Научились работать с точками останова и анализом использование ЦП.

В ходе выполнения заданий на учебную практику была произведена установка не обходимых приложений, таких как Linux, компилятор GCC, Radare2, Iaito для обратного проектирования.

Также мы выполнили обратное проектирования. Работали с файлами и компилировали через Терминал.

Мы выполнили дизассемблирование и работали с графическим интерфейсом Iaito, Radare2, и через них изменяли неправильный код.

# ЛИТЕРАТУРА

1. (бакалавриат), 38.03.05 (бакалавриат) и 10.05.02 (специалитет) всех профилей подготовки / Юрий Владимирович Ланских ; ВятГУ, ФАВТ, каф. АТ. - Киров: [б. и.], 2015. - 138 с.
2. 3.Золотов, С. Ю. Проектирование информационных систем [Электронный ресурс] / С.Ю. Золотов. - Томск: Эль Контент, 2013. - 88 с.
3. Карпенков, С. Х. Технические средства информационных технологий [Электронный ресурс] / С.Х. Карпенков. - 3-е изд., испр. и доп. - М.|Берлин: Директ-Медиа, 2015. - 376 с.
4. Коноплева, И. А. Информационные технологии [Электронный ресурс] / И.А. Коноплева. - 2-е изд., перераб. и доп. - Москва: Проспект, 2014. - 328 с.
5. Ланских, Юрий Владимирович Предметно-ориентированные информационные системы [Электронный ресурс] : учеб. пособие для студентов направления 09.03.02, 10.03.01,
6. Проектирование информационных систем. Лекция 1. Презентация [Электронный ресурс]. - Москва: Национальный Открытый Университет «ИНТУИТ», 2014. - 27 с.
7. Советов, Борис Яковлевич. Информационные технологии [Электронный ресурс]: учебник / Б. Я. Советов, В. В. Цехановский. - 6-е изд. - Москва: Юрайт, 2015. - х эл. опт. диск (CD-ROM)
8. Советов, Борис Яковлевич. Информационные технологии [Электронный ресурс]: учебник / Б. Я. Советов, В. В. Цехановский. - 6-е изд. - Москва: Юрайт, 2015. - х эл. опт. диск (CD-ROM)
9. Страбыкин, Дмитрий Алексеевич. Организация ЭВМ: лабораторный практикум на компьютерах: учеб. пособие для студентов направления подготовки 09.03.01 (230100.62) / Д. А. Страбыкин; ВятГУ, ФАВТ, каф. ЭВМ. - 3-е изд., перераб. и доп. - Киров: [б. и.], 2013. - 62 с.

# ПРИЛОЖЕНИЕ 1

**Программное обеспечение для работы**

|  |  |
| --- | --- |
|  |  |
| Рис. 1 VirtualBox | Рис. 2 Visual Studio |

# ПРИЛОЖЕНИЕ 2

**Работа с системой контроля версий**

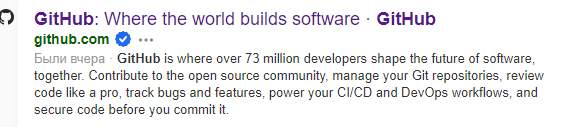


Рис. 1 Сайт GitHub

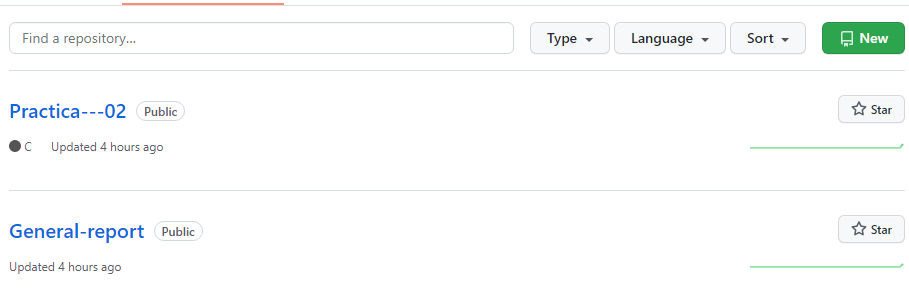


Рис. 2 Список репозиторий

Вписываем название проекта

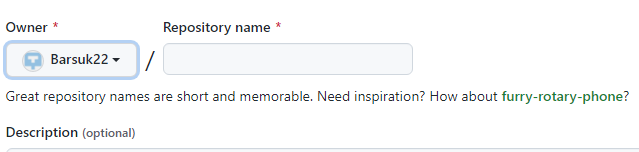


Рис. 3 Создание нового репозитория

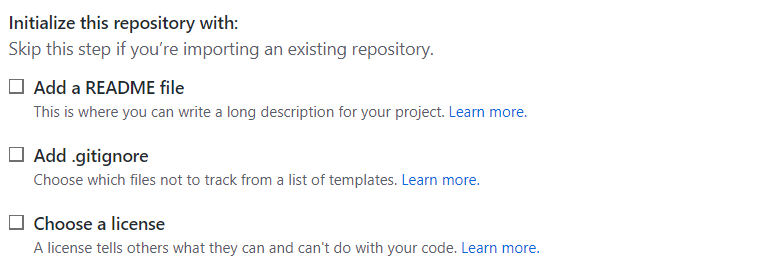


Рис. 4 Создание README file

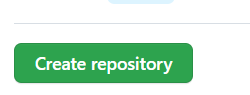


Рис. 5 Создание репозитория

Пишем информацию о проекте и разработчике в README file

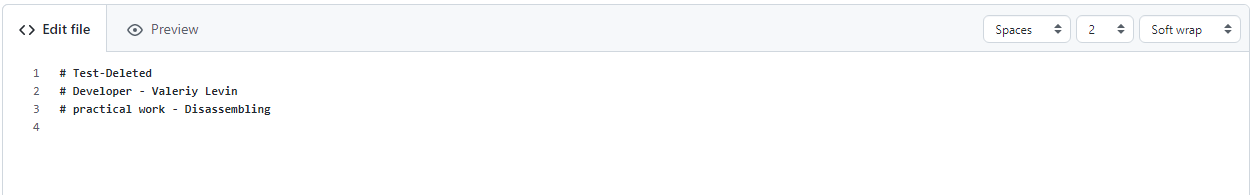


Рис. 6 Информация о разработчике и проекте

Нажимаем Commit Changes

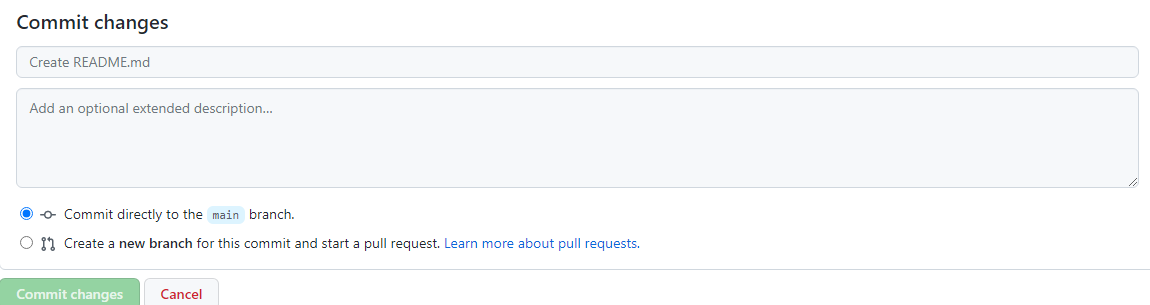


Рис. 7 Коммитим

Нажимаем Add file – Upload files

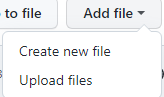


Рис. 8 Загружаем наши файлы

Закидываем нужные файлы

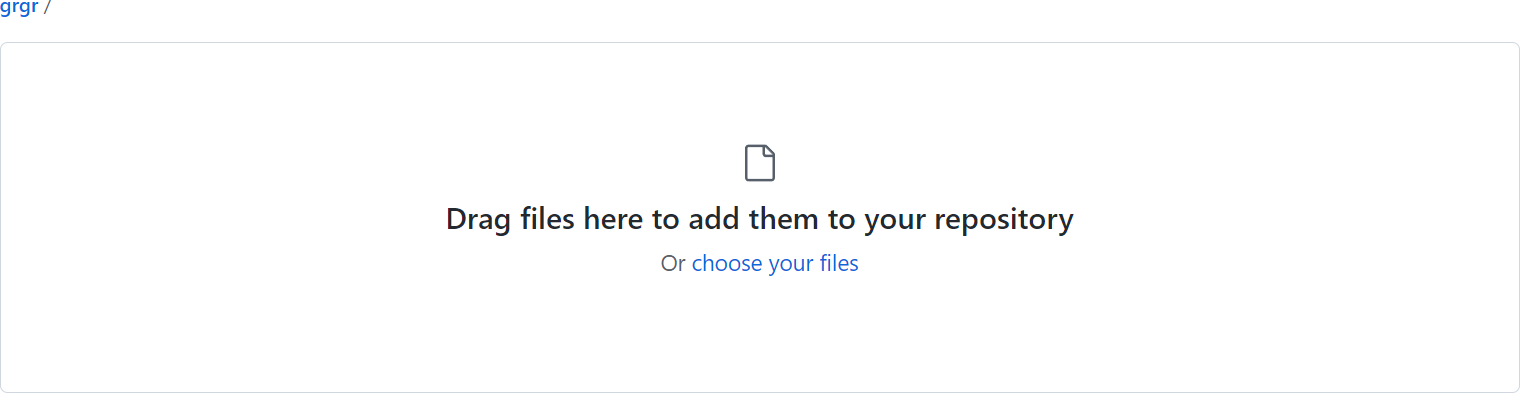


Рис. 9 Закидываем файлы

В окошко переносим файлы, которые хотим сохранить в проекте

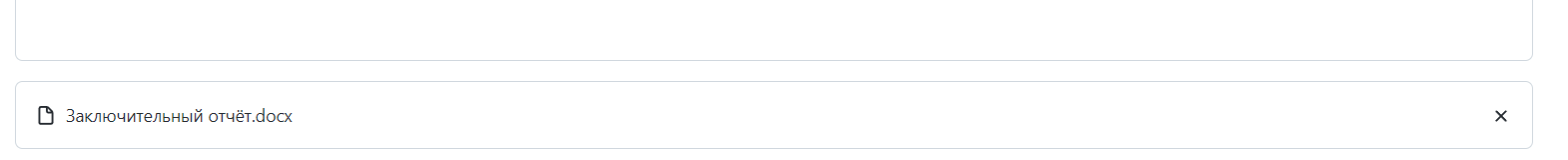


Рис 10 Перенесли файлы

Снова нажимаем на Commit changes

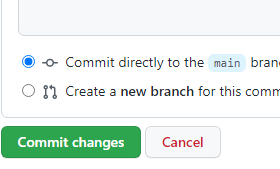


Рис. 11 Коммитим загруженные файлы

И готово. Все файлы за коммитены

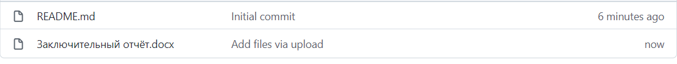


Рис. 12 Все файлы в репозитории

# ПРИЛОЖЕНИЕ 3

**Репозитории на GitHub**

<https://github.com/Barsuk22/MDK-03.02> - Изменение производительности приложения посредством анализа использования ЦП

[https://github.com/Barsuk22/Practica---02](https://github.com/Barsuk22/Practica---02%20) - Обратное проектирование и дизассемблирование.

# ПРИЛОЖЕНИЕ 4



Рис. 1

Нажимаем на кнопку «создать»

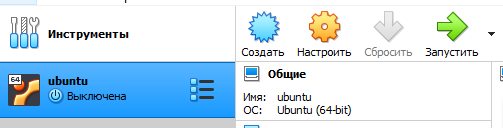


Рис. 2 Рабочий стол виртуальной машины

Даем имя, выбираем тип и версию ОС, и нажимаем далее

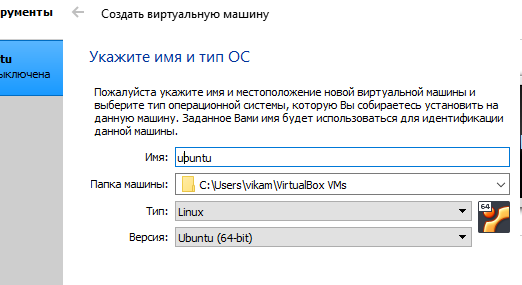


Рис. 3 Даем имя

Устанавливаем объем памяти, необходимый для комфортной работы

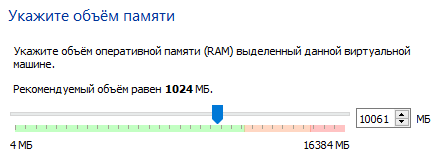


Рис. 4 Выбираем объем памяти

Выбираем «Создать новый виртуальный жесткий диск»

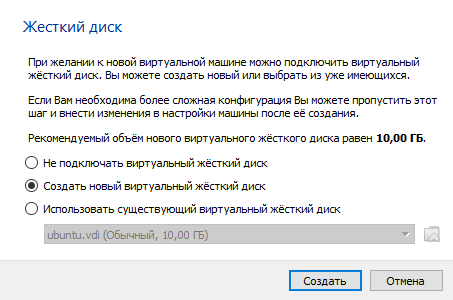


Рис. 5 Создали новый виртуальный жесткий диск

Указываем тип VDI

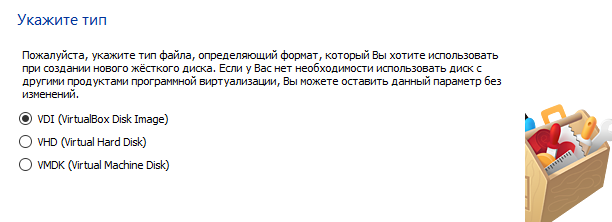


Рис. 6 Выбираем VDI

Выбираем динамический жёсткий диск

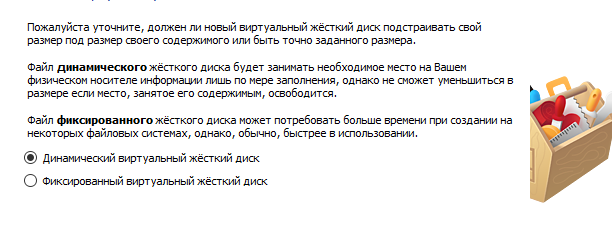


Рис. 7 Выбираем динамический жесткий диск

Выбираем объем памяти для удобной работы

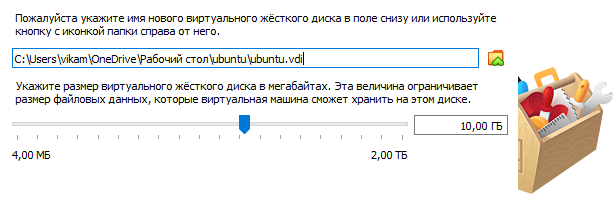


Рис. 8 Объем памяти

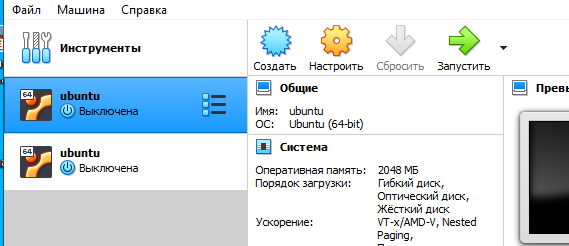


Рис. 9 Виртуальная машина установлена