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.К.Муратова

\_\_\_\_\_\_\_\_\_\_\_дата

2021-2022 уч.г.

Содержание

[Введение 3](#_Toc88660589)

[1.Список терминов и определений 5](#_Toc88660590)

[2.Функциональные требования 6](#_Toc88660591)

[3. Измерение производительности приложения посредством анализа использования ЦП 6](#_Toc88660592)

[3.1. Шаг 2 Анализ данных о загрузке ЦП 9](#_Toc88660593)

[3.2. Просмотр внешнего кода 11](#_Toc88660594)

[4. Установка ПО 12](#_Toc88660595)

[4.1. Создание исходного файла 12](#_Toc88660596)

[5. Обратное проектирование 14](#_Toc88660597)

[6.Дизассемблирование 16](#_Toc88660598)

[ЗАКЛЮЧЕНИЕ 20](#_Toc88660599)

[ЛИТЕРАТУРА 21](#_Toc88660600)

# 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 ПО, 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 ПО (Linux, компилятор GCC, radare2, iaito) для обратного проектирования,
* выполнить обратное проектирование,
* выполнить дизассемблирование.

# Список терминов и определений

***Репозиторий***(от англ. **repository** – склад, хранилище; встречается также написание «репозитарий») – это профессиональный участник рынка ценных бумаг, осуществляющий ведение реестров договоров РЕПО и договоров с деривативами, заключенных на организованных торгах.

***Дизассемблер*** —это компьютерная программа, которая переводит машинный язык на язык ассемблера-операция, обратная операции ассемблера.

***GitHub*** — это платформа, хранящая различные Git-репозитории на своих многочисленных серверах. Также GitHub называют крупнейшим веб-сервисом для хостинга и совместной разработки IT-проектов.

В программировании ***то́чка остано́ва*** (англ. breakpoint) — это преднамеренное прерывание выполнения программы, при котором выполняется вызов отладчика (одновременно с этим программа сама может использовать точки останова для своих нужд).

***Radare2*** (также известный как r2)-это полный фреймворк для обратного проектирования и анализа двоичных файлов, состоящий из набора небольших утилит, которые могут использоваться вместе или независимо от командной строки.

***Компилятор*** - это программа, которая выполняет преобразование текста программы в другое представление, обычно машинный код, без его запуска, статически.

***GCC***- это свободно доступный оптимизирующий компилятор для языков C, C++, Ada 95, а также Objective C. Его версии применяются для различных реализаций Unix (а также VMS, OS/2 и других систем PC), и позволяют генерировать код для множества процессоров.

***Програ́ммное обеспе́чение*** (допустимо также произношение обеспече́ние) (ПО) — программа или множество программ, используемых для управления компьютером (ISO/IEC 26514:2008).

***Linux*** — семейство Unix-подобных операционных систем на базе ядра Linux, включающих тот или иной набор утилит и программ проекта GNU, и, возможно, другие компоненты.

***Обратная разработка*** — 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 с аналогичными функциями, но без прямого копирования.

# Функциональные требования

Для выполнения поставленных задач, мы использовали ПО Linux, в нём мы устанавливали пакеты

# 3. Измерение производительности приложения посредством анализа использования ЦП

1. Открыли проект и установили точки остановы, где хотим проверить загрузку ЦП.

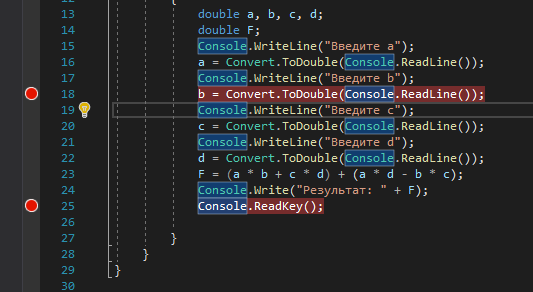


Рис. 1 (Точка остановы)

1. Ставим вторую точку остановы в конце функции

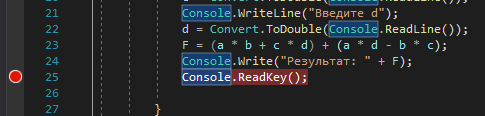


Рис. 2 (Точка остановы в конце функции)

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

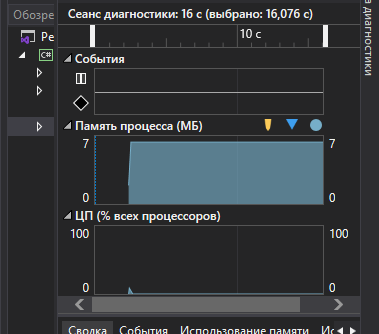


Рис. 3 (Окно диагностики)

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**.

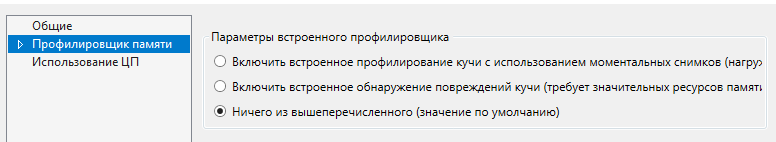


Рис. 4 (Профилировщик памяти)

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



Рис. 5 (Отладка)

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

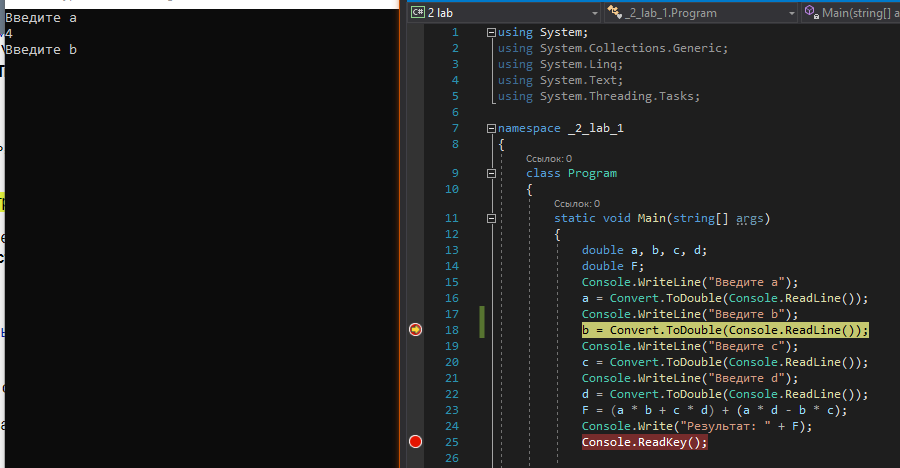


Рис. 6 (Первая точка остановы)

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

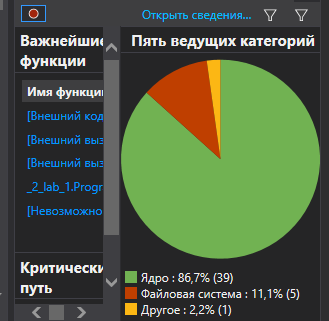


Рис. 7 (Загрузка ЦП)

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.

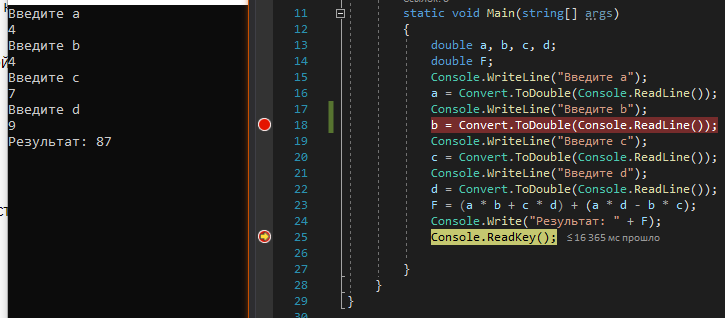


Рис. 8 (Точка остановы в конце функции)

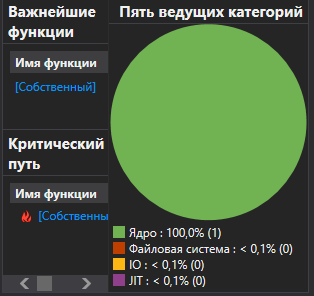


Рис. 9 (Загрузка ЦП)

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

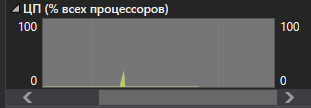


Рис. 10 (Временная шкала ЦП)

### . Шаг 2 Анализ данных о загрузке ЦП

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

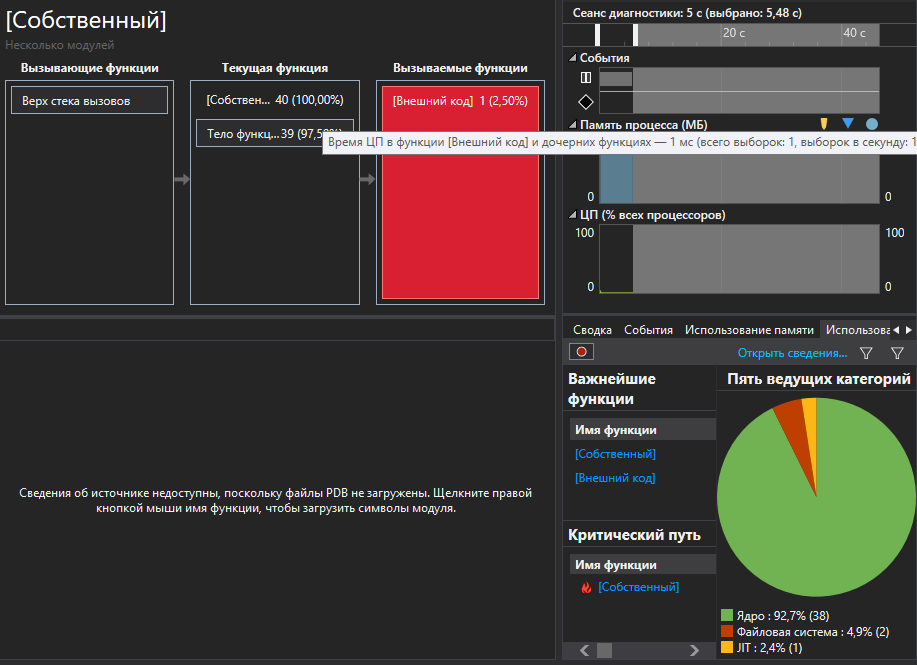


Рис. 11 (Собственный)

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.**

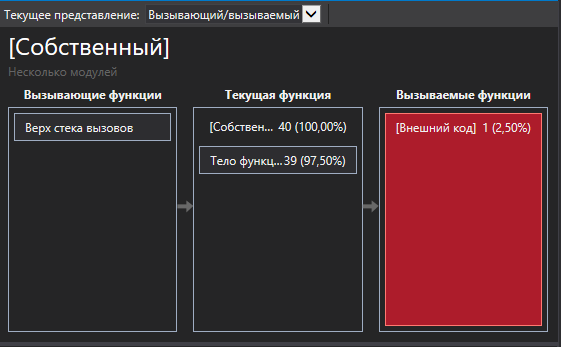


Рис. 12 (Вызывающий/вызываемый)

В этом представлении выбранная функции отображается в заголовке и в поле Текущая функция (в этом примере GetNumber). 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. (Можно выбрать любое поле, чтобы изменить текущую функцию.)

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.

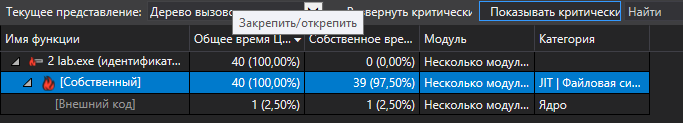


Рис. 13 (Дерево вызовов)

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



Рис. 14 (Критический путь)

**3.2. 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.

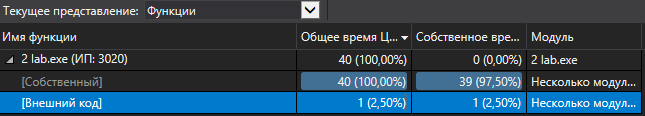


Рис. 15 (Внешний код)

# 4. Установка ПО

### 4.1. Создание исходного файла

1. Устанавливаем компилятор GCC и все необходимые для него компоненты

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



Рис. 16 (Установка компилятора GCC)

И пойдёт установка

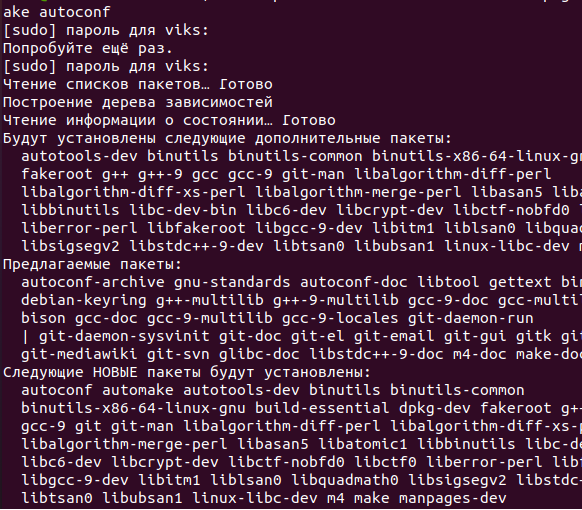


Рис. 17 (Процесс установки)

1. Устанавливаем необходимые пакеты для работы с radare2

sudo apt install git build-essential make

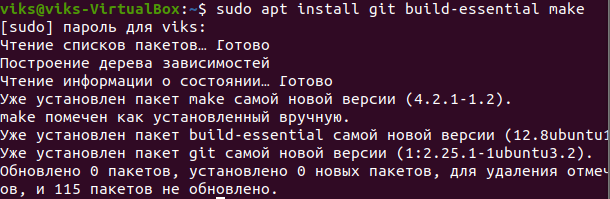


Рис. 18 (Пакеты radare2)

1. Установка radare2:

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

radare2/sys/install.sh

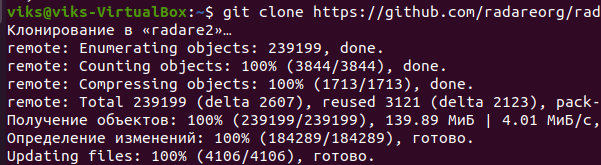


Рис. 8 (Установка radare2)

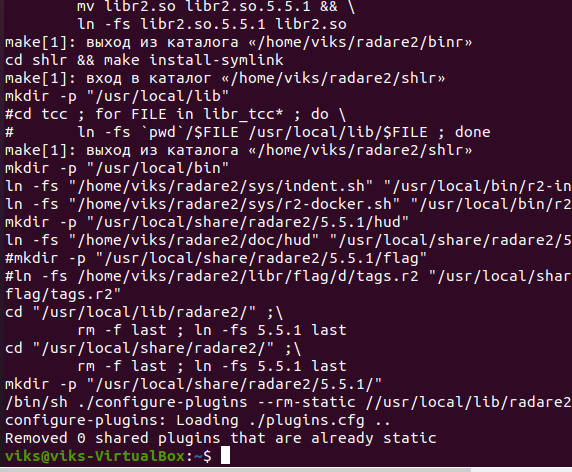


Рис. 19 (Процесс установки)

1. Проверяем и обновляем radare2

r2pm update



Рис. 20 (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

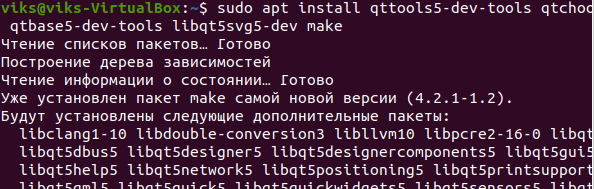


Рис. 21 (Установка необходимых компонентов)

1. Повторить

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

r2pm install iaito



Рис. 22 (Установка iaito)

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

r2pm install r2ghidra



Рис. 23 (Установка декомпилятора)

# 5. Обратное проектирование

1. Создаем файл с кодом для компиляции 101

touch program.с



Рис. 24 (Создаём файл)

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 < 4){  
printf ("Дизессамблирование\n");

x = x + 1;  
}  
return 0;  
}

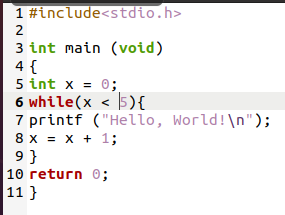


Рис. 25 (Программа)

1. Компилируем

gcc program.c -o program



Рис. 26 (Компиляция)

1. Открываем и проверяем наш файл

./program

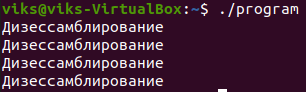


Рис. 27 (Проверка файла)

1. Открываем папку и открываем файл



Рис. 28 (Открываем папку)

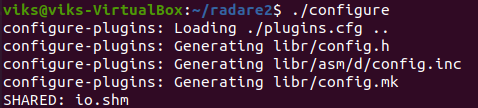


Рис. 29 (Открываем файл)

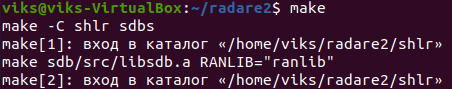


Рис. 30

# Дизассемблирование

1. Запускаем «Iaito»

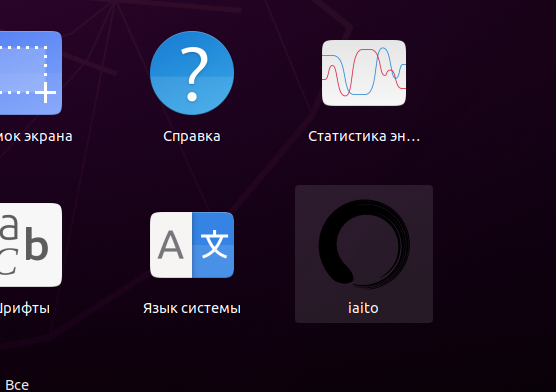


Рис. 31 (Приложение Iaito)

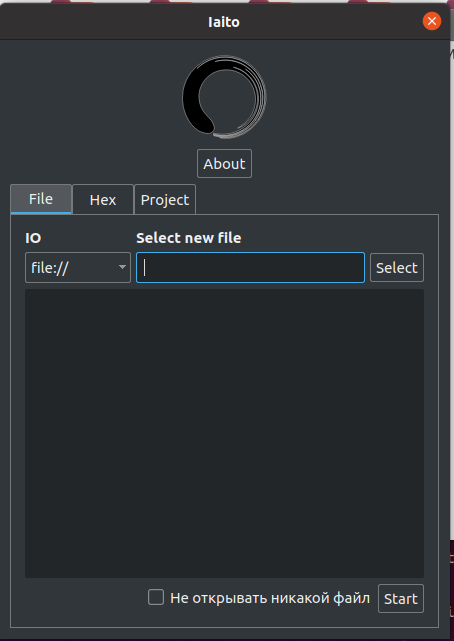


Рис. 32 (Графический интерфей)

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

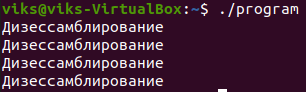
****

Рис. 33 (Проверка файла)

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

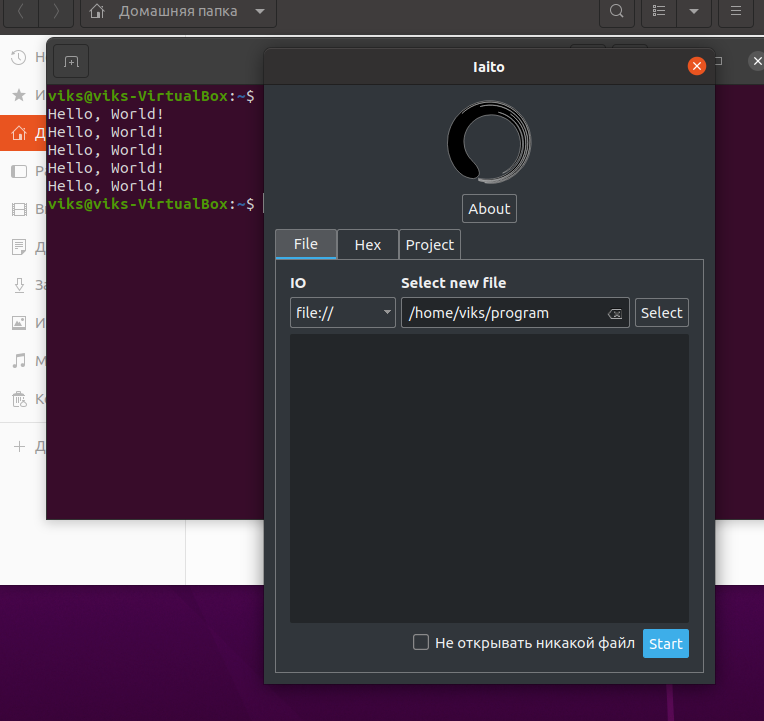


Рис. 34 (Открываем файл)

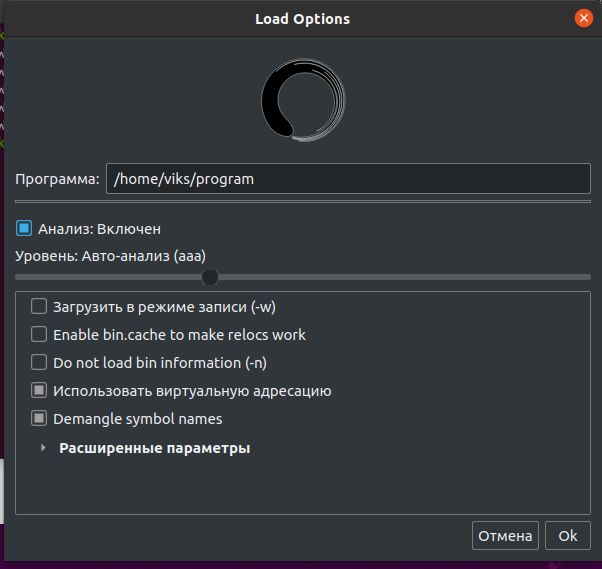


Рис. 35 (Настройки)

1. Открываем слева «main»

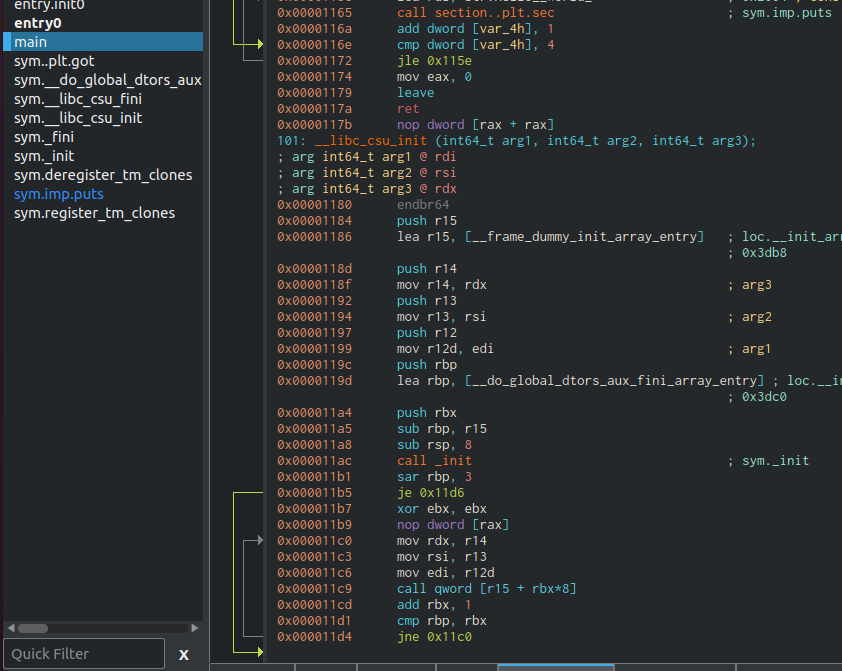


Рис. 36 (Открываем слева «main»)

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

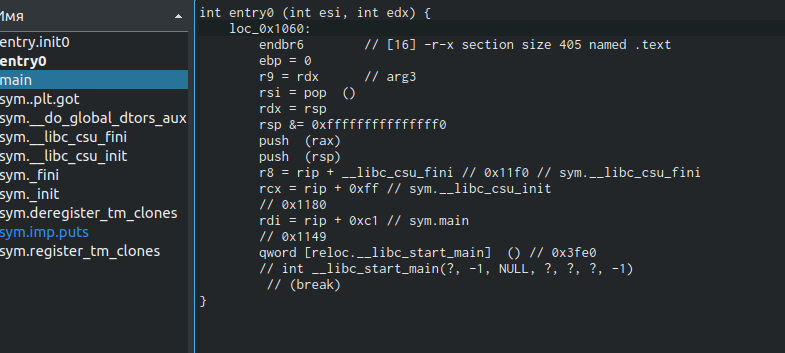


Рис. 37 (Декомпилятор)

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

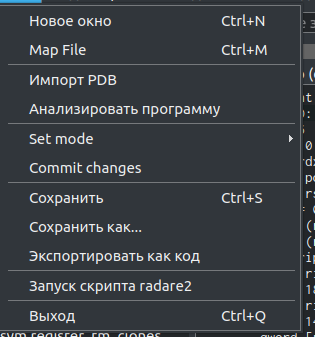


Рис. 38 (Cache mode)

1. Так как фраза «Hello world!» выводится 5 раз, а нам необходимо увеличить вывод до 10 раз, то смотря на код, можно увидеть, что используется цикл «while» по значение переменной [var4\_h] равной 4 (т.к While <=4, а цикл начинается с 0). Выделяем цифру 4 и нажимаем «Edit» меняя значение переменное с 4 до 10.



Рис. 39 (Переменная var4\_h)

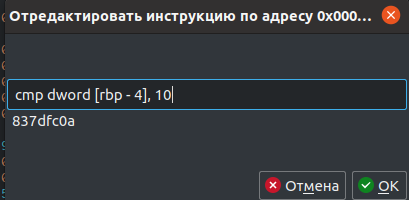


Рис. 40 (Изменение Цикла)

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

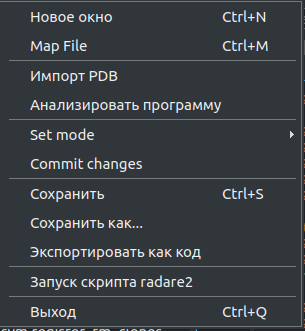


Рис. 41 (Commit changes)

1. Запускаем наш файл для проверки

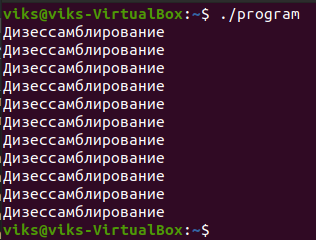


Рис. 42 (Дизессамблирование)

# ЗАКЛЮЧЕНИЕ

Целью учебной практики по ПМ.03 «Ревьюирирование программных продуктов» являлось, научиться выполнять прямое и обратное проектирование ПО, 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:

1. Изучили термины (репозиторий, дизассемблер, GitHub, radare2, компилятор, GCC, ПО, обратная разработка, Linux)
2. измерить производительность приложения посредством анализа использования ЦП,
3. установить ПО (Linux, компилятор GCC, radare2, iaito) для обратного проектирования,
4. выполнить обратное проектирование,
5. выполнить дизассемблирование.

<https://github.com/viks17/praktika.git>

ПРИЛОЖЕНИЕ 6

# 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. Корячко, В. П. Процессы и задачи управления проектами информационных систем [Электронный ресурс] / В.П. Корячко. - Москва: Горячая линия - Телеком, 2014. - 376 с.
6. Ланских, Юрий Владимирович Предметно-ориентированные информационные системы [Электронный ресурс] : учеб. пособие для студентов направления 09.03.02, 10.03.01,
7. Проектирование информационных систем. Лекция 1. Презентация [Электронный ресурс]. - Москва: Национальный Открытый Университет «ИНТУИТ», 2014. - 27 с.
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. 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)
10. 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.