Санкт–Петербургский государственный университет

Царёв Никита Евгеньевич

Выпускная квалификационная работа

Разработка обучающего веб-инструмента удаленной сборки и интерактивной отладки программ

Уровень образования: бакалавриат
Направление 01.03.02 «Прикладная математика и информатика»
Основная образовательная программа СВ.5005.2018 «Прикладная математика, фундаментальная информатика и программирование»
Профиль «Современное программирование»

Научный руководитель:
профессор факультета МКН СПбГУ, д.ф.-м.н.
А. С. Куликов

Консультант:
Ассистент кафедры МОЭВМ СПбГЭТУ ЛЭТИ,
К. В. Чайка

Рецензент:
доцент НовГУ, к. т. н. П. М. Довгалюк

Санкт-Петербург
2022 г.

Содержание

Введение

Обучение языку ассемблера и архитектуре ЭВМ являются важной составляющей большинства учебных программ по подготовке программистов. Эти курсы задают основу, необходимую для освоения других дисциплин, таких как разработка компиляторов, устройство операционных систем, системное программирование и многих других. Современные инструменты для разработки на языке ассемблера расчитаны на профессионалов в этой области и требуют многих смежных знаний для их продуктивного использования. К таким знаниям, например, относится опыт работы с командной строкой, понимание механизмов компиляции и компоновки программ и умение работать с консольными отладчиками. Всё это создаёт препятствия тем, кто только начинает свой профессиональный путь и не может с лёгкостью использовать и настраивать подобные инструменты, имеющие высокий порог вхождения.

Преподаватели и организаторы процесса обучения языку ассемблера также сталкиваются с похожими трудностями. Инструменты, используемые в процессе обучения, должны быть максимально наглядными, а проверка заданий — как можно более автоматизированной. Для наглядности, инструмент должен предоставлять удобный способ просматривать внутреннее состояние процессора, на котором запускается пользовательский код, а также выполнять различные манипуляции над ним, например, выполнение по шагам или редактирование значений регистров. В этой работе рассматривается процесс реализации программного инструмента удалённой сборки и отладки программ на языке ассемблера, призванного решить эти проблемы.

В первой главе данной работы подробно рассматриваются и анализируются многие существующие альтернативные решения проблемы. Исходя из этого анализа, строятся требования к разрабатываемому решению, и, во второй главе, приводится описание архитектуры программной реализации разработанного инструмента. В третьей главе производится исследование свойств разработанного решения, а также поиск его возможных будущих улучшений.

Постановка задачи

Цель данной работы состоит в разработке обучающего веб-инструмента удалённого запуска, отладки и проверки программ на языке ассемблера.

Задачи данной работы:

1. Исследование существующих решений для запуска и отладки программ на языке ассемблера, а также решений для обучения языку ассемблера.
2. Формирование требований к разрабатываемому инструменту.
3. Разработка программной архитектуры инструмента и его реализация.
4. Исследование свойств решения.

Объектом исследования являются системы запуска и отладки программ на языке ассемблера.

Предметом исследования является наглядность и удобство использования таких систем в учебном процессе.

Практическая ценность работы состоит в том, что разработанный инструмент позволит проводить обучение языку ассемблера более наглядно для студентов.

Глава 1. Обзор предметной области

1.1 Критерии сравнения и отбора аналогов

В мире существует множество решений для запуска ассемблерного кода, а также решений для обучения языку ассемблера. В этой главе рассматривается несколько таких решений, каждое из них анализируется в контексте следующих **критериев сравнения**:

1. Поддержка запуска ассемблерного кода на разных диалектах и на разных архитектурах.
2. Поддержка отладки: выполнение по шагам, поддержка точек останова, редактирования регистров/памяти, визуализация стека вызовов.
3. Поддержка задач и их автоматической проверки.
4. Поддержка интеграции с системами управления обучением.
5. Возможность работы без установки дополнительного программного обеспечения на устройстве пользователя.
6. Возможность самостоятельной установки и развёртывания системы на выделенном сервере, доступность исходного кода.

Альтернативные решения искались по запросам “online assembly debugger”, “online assembly ide”, “educational assembly ide”. Также были рассмотрены популярные решения для организации учебного процесса при использовании других языков программирования.

1.2 Существующие решения

1.2.1 Ideone

Ideone[17] является онлайн компилятором и средой разработки, поддерживающей более 60 языков программирования, в том числе несколько диалектов ассемблера.

Поддерживается запуск ассемблерного кода на архитектурах x86 (Intel и AT&T диалекты) и x86-64 (только Intel диалект). Отладка не поддерживается. Поддержки задач, их автоматической проверки нет, соответственно нет и интеграции в системы управления обучением.

Взаимодействие с системой происходит через веб-интерфейс, установки дополнительного ПО не требуется. Система имеет закрытый исходный код, самостоятельно установить систему на выделенный сервер не представляется возможным.

1.2.2 OneCompiler

OneCompiler[19] является онлайн компилятором и средой разработки, поддерживающей, в том числе, и Intel диалект x86 ассемблера.

Поддержиывается запуск кода, есть возможность указать содержимое стандартного потока ввода перед запуском. Отладка не поддерживается. Поддержки задач, их автоматической проверки нет, соответственно нет и интеграции в системы управления обучением.

Взаимодействие с системой происходит через веб-интерфейс, установки дополнительного ПО не требуется. Система имеет закрытый исходный код, самостоятельно установить систему на выделенный сервер не представляется возможным.

1.2.3 ASM Debugger

ASM Debugger[3] является инструментом для пошаговой отладки простых программ на языке ассемблера. Особенностью инструмента является то, что он не использует запуск программ на реальном аппаратном обеспечении. Вместо этого, на языке Javascript реализовано подмножество инструкций x86 ассемблера.

Поддерживается запуск ассемблерного кода на архитектуре x86 с Intel диалектом. Поддерживается пошаговое исполнение, просмотр значений регистров. Поддержки задач, их автоматической проверки нет, соответственно нет и интеграции в системы управления обучением.

Взаимодействие с инструменетом происходит через веб-интерфейс, установки дополнительного ПО не требуется. Инструмент имеет открытый исходный код, установки серверной части не требуется, так как вся логика инструмента исполняется в браузере клиента.

1.2.4 Davis

Davis[5] является инструментом для запуска и пошагового исполнения простых программ на языке ассемблера. Как и ASM Debugger, данный инструмент включает в себя реализацию эмулятора x86 ассемблера, использующего диалект Intel. Эмулятор написан на языке TypeScript и реализует основные арифметические инструкции, условные и безусловные переходы, инструкции работы со стеком. Также поддерживается инструкция INT, позволяющая выполнять вывод на экран.

Инструмент предоставляет возможность пошагового исполнения, установки точек останова, просмотра содержимого регистров и памяти. Можно настроить частоту исполнения инструкций в режиме исполнения: от 100 до 500 миллисекунд.

Поддержки задач, их автоматической проверки и интеграции в системы управления обучением нет. Также как и ASM Debugger, данный инструмент доступен через веб-интерфейс, причём серверная часть отсутствует. Инструмент имеет открытый исходный код.

1.2.5 OnlineGDB

OnlineGDB[23] является онлайн компилятором, отладчиком и средой программирования, поддерживающим более 20 различных языков программирования. Для многих из них, например, для C, C++ и языка ассемблера, поддерживается интерактивная отладка. При работе с этими языками создаётся интерактивная сессия в отладчике GDB.

Поддерживается запуск ассемблерных программ на архитектуре x86-64 с AT&T диалектом. Программе можно передать аргументы командной строки, а также взаимодействовать с ней через стандартные потоки ввода и вывода. Поддерживается запуск в режиме отладки, позволяющий устанав-

ливать точки останова, просматривать информацию о состоянии регистров, стека вызовов, а также следить за произвольными выражениями. Инструмент позволяет исполнять программу по шагам, до выхода из функции, или до следующей точки останова. Также можно выполнять произвольные команды отладчика GDB.

Существует поддержка создания учебных классов из веб-интерфейса. В них можно добавлять преподавателей и студентов, а также публиковать задания. В заданиях можно задавать режим проверки: ручной или автоматический. В автоматическом режиме проверки программа запускается на наборе тестов. Каждый тест представляет из себя входные данные и эталонный ответ. При запуске программы, входные данные записываются ей на стандартный поток ввода, а вывод программы сравнивается с эталонным ответом. Также для задания можно указать шаблон кода, который будет показан студенту вместо пустого окна редактора перед выполнением работы. Интеграции с существующими системами управления обучением нет, но существует возможность просмотра результатов выполнения задания в виде таблицы.

Инструмент имеет закрытый исходный код, установка его на выделенный сервер не представляется возможным.

1.2.6 SASM

SASM[20] представляет из себя кроссплатформенную среду разработки на языке ассемблера для архитектур x86 и x86-64 с использованием ассемблеров NASM, GNU Assembler, FASM, MASM. Поддерживаются диалаекты Intel и AT&T.

Поддерживается запуск ассемблерного кода, поддерживается выполнение по шагам, точки останова, просмотр и редактирование регистров и памяти, а также произвольные команды GDB.

Поддержки задач, их автоматической проверки нет, соответственно нет и интеграции в системы управления обучением.

Для использования инструмента необходима его установка на компьютер пользователя. Инструмент имеет открытый исходный код.

1.2.7 YASP

YASP[7] является онлайн средой разработки для вымышленного микроконтроллера, включающей в себя ассемблер, отладчик и эмулятор. Диалект ассемблера для этого вымышленного микроконтроллера был придумал авторами для упрощения его преподавания. По словам авторов, цель проекта заключается в создании среды, используя которую, студенты могут выучить язык ассемблера, чтобы лучше понимать внутреннее устройство компьютеров [7].

Поддерживается запуск ассемблерных программ на этой вымышленной архитектуре. Также поддерживается режим отладки, в котором можно видеть текущее состояние эмулируемого микроконтроллера. Например, при исполнении определённых команд, можно видеть, как загораются или потухают светодиоды, нарисованные поверх фотографии микроконтроллера. Также поддерживается просмотр состояния оперативной памяти, неизменяемой памяти и регистров. Код можно исполнять как по шагам, так и до следующей точки останова, пользуясь соответствующими кнопками в пользовательском интерфейсе.

Несмотря на то, что инструмент предназначен для обучения языку ассемблера, поддержки задач, их проверки и интеграции с системами управления обучением нет. Инструмент имеет открытый исходный код и может быть использован через веб-браузер, без необходимости его установки на компьютер пользователя.

1.2.8 JetBrains Clion + EduTools

Clion[32] — это интегрированная среда разработки от компании JetBrains, предназначенная, в первую очередь, для разработки приложений на языках C и C++. Язык ассемблера не поддерживается ни в каком виде, но существуют сторонние плагины, которые решают эту проблему, например, NASM Assembly Language[15].

Компиляция и запуск кода на языке ассемблера возможны, если модифицировать должным образом файлы системы описания сборки CMake. Отладка ассемблерного кода не поддерживается.

Плагин EduTools[33] позволяет создавать и писать задачи с автоматическими тестами, что упрощает проверку решений. Отсутствует поддержка задач с закрытыми (недоступными для обучающегося) тестами. Интеграция с системами управления обучением отсутствует.

Для использования данной среды разработки необходима её установка на компьютер пользователя. Она имеет закрытый исходный код.

1.2.9 GitHub Classroom + Visual Studio Code

GitHub Classroom[13] — это сервис, позволяющий давать учебные задания в виде git-репозиториев. GitHub Classroom предоставляет возможность добавить кнопку «открыть в Visual Studio Code»[14], которая позволяет открыть репозиторий с преднастроенными плагинами в этом редакторе.

Для того, чтобы настроить поддержку языка ассемблера в Visual Studio Code, требуется установка дополнительных плагинов. Также преподавателю в шаблонном репозитории необходимо будет настроить компиляцию и запуск в файлах `tasks.json` и `launch.json`. Отладка не поддерживается.

GitHub Classroom позволяет добавлять тесты через веб-интерфейс преподавателя. В качестве теста может выступать набор входных данных и эталонных ответов к ним, так и путь до сценария для автоматической проверки. В первом случае входные данные передаются программе через стандартный поток ввода, а вывод программы сравнивается с эталонным ответом.

На компьютер пользователя необходимо устанавливать Visual Studio Code, компилятор и отладчик. GitHub Classroom имеет закрытый исходный код, установить свою копию на выделенный сервер не представляется возможным.

1.2.10 Stepik

В системе управления обучением Stepik[39] есть режим задания Code Challenge, который позволяет проверять код, написанный на различных языках программирования. Поддерживается Intel диалект x86 и x86-64 ассемблера. Отладка запускаемого кода не поддерживается.

Поддерживаются задачи и их автоматическая проверка на скрытых тестах. Тесты должны иметь вид набора входных данных и эталонных ответов. Входные данные передаются программе через стандартный поток ввода, а вывод программы сравнивается с эталонным ответом. Взаимодействие с системой происходит через веб-интерфейс, установка дополнительного ПО не требуется. Система имеет закрытый исходный код.

1.2.11 Moodle + Virtual Programming Lab

Для системы управления обучением Moodle[1] существует плагин Virtual Programming Lab[30], котторый позволяет запускать и проверять код, написанный на различных языках программирования.

Поддерживается Intel диалект x86 ассемблера, отладка не поддерживается. Поддерживаются задачи и их автоматическая проверка на скрытых тестах. Тесты должны иметь вид набора входных данных и эталонных ответов. Входные данные передаются программе через стандартный поток ввода, а вывод программы сравнивается с эталонным ответом.

Взаимодействие с системой происходит через веб-интерфейс, установка дополнительного ПО не требуется. И система Moodle и плагин Virtual Programming Lab имеют открытый исходный код. Соответственно, есть возможность установки этой связки на выделенный сервер.

1.2.12 Git репозиторий с задачами

Одним из популярных способов организовать проверку заданий при проведении курсов по программированию является специально организованный git репозиторий. К примеру может ожидаться, что студенты выполняют задания в своих локальных копиях, а задания проверяются специально написанными сценариями, расположенными в том же репозитории.

При таком подходе, учащиеся могут выбирать любой удобный им редактор кода. Для запуска кода могут быть предоставлены готовые утилиты. То же самое нельзя сказать об отладке, скорее всего студенту придётся познакомиться с GDB или другой подобной утилитой.

Такой подход применяется в учебном процессе на практике. Возможно

автоматизировать и генерацию репозиториев с задачами для каждого пользователя, и закрытые тесты, и даже интеграцию с системами управления обучением. На практике таким мало кто занимается.

Такой подход требует от орагнизаторов курса реализовать сценарии запуска и проверки решений. От студентов же требуется установка дополнительного программного обеспечения (такого как git, компилятор, отладчик) и наличие навыков работы с инструментами командной строки. Также может требоваться использование конкретной операционной системы, так как зачастую такие сценарии расчитывают на наличие конкретного пользовательского окружения.

1.3 Сравнение существующих решений

Решение	Архитектура	Отладка	Учебный процесс	Без установки	Открытый исходный код
Ideone	x86, x86-64			✓	
OneCompiler	x86			✓	
ASM Debugger	x86 (часть)	✓		✓	✓
Davis	x86 (часть)	✓		✓	✓
OnlineGDB	x86-64	✓	✓	✓	
SASM	x86, x86-64	✓			✓
YASP	своя	✓		✓	✓
Clion + EduTools	x86 (плагин)		частично		
GitHub + VSCode	зависит		частично		
Stepik	x86, x86-64		✓	✓	
Moodle + VPL	x86		✓	✓	✓
Git репозиторий	зависит		частично		

Таблица 1: Сравнение существующих решений

Краткая информация о решениях представлена в таблице 1. Среди расмотренных альтернатив можно выделить несколько групп схожих решений.

Первой такой группой являются онлайн компиляторы. К ним можно отнести Ideone и OneCompiler. Они представляют веб редакторы с возможностью компиляции и запуска кода на разных языках. В этих системах языку ассемблера не уделяется особого внимания, так как основная масса пользователей таких систем использует их для написания кода на высокоуровневых языках. Также эти системы имеют закрытый исходный код.

В качестве второй группы можно выделить такие системы как Stepik, Moodle и JetBrains EduTools. Они предназначены, в первую очередь, для образовательных процессов. Поддержка задач, направленных на иизучение языка ассемблера, не является их основной целью. Так, для системы Moodle, требуется сторонний плагин, а среды разработки JetBrains поддерживают механизм отладки многих языков программирования, но не ассемблера.

В качестве ещё одной группы решений можно выделить ASM Debugger, Davis и YASP. Они представляют из себя простые веб-инструменты для запуска и пошагового исполнения ассемблерного кода. Эти инструменты реализуют свой диалект ассемблера и исполняются не на реальном процессоре, а напрямую в браузере пользователя. Помимо отсутствия поддержки запуска на настоящих машинах, у этих инструментов есть ещё и другой недостаток: отсутствие возможности интеграции в процесс обучения из-за того, что они исполняются исключительно на стороне пользователя.

Также хочется выделить решения, требующие сложной настройки со стороны преподавателей, такие как интеграция GitHub Classroom с Visual Studio Code и использование git репозитория для организации учебного процесса. Эти решения отличаются гибкостью, но требуют написания сценариев и конфигурационных файлов перед использованием. Также они требуют установки дополнительного программного обеспечения на компьютерах студентов.

В качестве последней группы можно рассмотреть такие инструменты, как SASM и OnlineGDB. Они имеют полную поддержку запуска и отладки ассемблерного кода, и используют настоящие компиляторы и отладчики для этого. OnlineGDB также отличается другими выгодными свойствами: он работает через веб-интерфейс, а также предоставляет определённые возможности для создания учебных комнат, заданий и их автоматической проверки.

К сожалению, этот инструмент обладает и недостатками: отсутствие возможности установить серверную часть на выделенный сервер из-за закрытогоо исходного кода. Также, несмотря на возможность создания заданий для студентов, в обоих этих инструментах отсутствует интеграция с другими системами управления обучением, что означает необходимость синхронизировать данные о пользователях и оценках между этими системами вручную.

1.4 Выводы

В этой главе были рассмотрены различные решения для запуска и отладки программ на языке ассемблера, а также решения, предназначенные для обучения языку ассемблера. Почти все рассмотренные решения по умолчанию работают только с архитектурами x86 и x86-64, только YASP использует другую, вымышленную, архитектуру. Во всех рассмотренных инструментах не хватает какой-либо важной функциональности. Большинство инструментов, имеющих функционал отладки ассемблерного кода, не поддерживают интеграцию в учебный процесс, а системы управления обучением не поддерживают отладку ассемблерного кода. Некоторые решения требуют явной установки на компьютер пользователя, что влечёт за собой проблемы с совместимостью и удобством использования. Также можно отметить, что значительная часть рассмотренных альтернатив являются проприетарными решениями. В частности, нельзя гарантировать доступность проприетарных веб-инструментов в течение длительного времени.

Глава 2. Реализация инструмента

2.1 Формулировка требований к решению

Наиболее важными характеристиками разрабатываемого инструмента будут те, которые выгодно его отличают от существующих аналогов. Исходя из этого, были сформулированы следующие функциональные требования к инструменту:

1. Инструмент должен быть доступен через веб-интерфейс и не требовать установки дополнительного программного обеспечения на устройстве пользователя.

2. Инструмент должен содержать возможность аутентификации студентов и преподавателей по протоколу LTI, а учащиеся должны уметь получать задания по конкретным задачам через системы управления обучением.

3. На странице задания для учащегося должно быть доступно условие задачи, редактор кода, возможность отправить решение на проверку и информация о предыдущих попытках решения.

4. Вместе с редактором кода должна быть доступна функциональность отладки: добавление и удаление точек останова, запуск, остановка, пошаговое исполнение программы, должен быть доступен просмотр и редактирование регистров процессора, а также вычисление произвольных выражений.

5. Запускаемые пользовательские программы должны быть ограничены по времени и используемой памяти, им должен быть запрещен доступ к файловой системе, сети, процессам и другим ресурсам операционной системы.

6. Для ассемблерного кода должны поддерживаться архитектуры x86-64 и AVR.

Также были сформулированы некоторые нефункциональные требования. Они ограничивают объём решаемой проблемы, а также задают свойства

системы, необходимые для её успешного применения в учебном процессе. Нефункциональные требования приведены ниже.

1. Инструмент, в первую очередь, предназначается для задач, направленных непосредственно на изучение языка ассемблера. Таким образом, архитектура проверки должна быть такой, чтобы решения задач могли исполняться в контексте непривилегированных процессов пользовательского пространства, без доступа к конкретным системным вызовам и периферии.

2. Инструмент должен быть расширяемым, должна быть возможность добавлять поддержку новых архитектур процессоров.

3. Дополнительное использование инструментом процессорного времени и оперативной памяти сервера должно быть сопоставимо с использованием аналогичных ресурсов отладчиком GDB.

4. Задержка реакции интерактивного отладчика на основные действия пользователя без учёта задержки сети должна быть как можно более незаметна для пользователя, то есть не превосходить десятков миллисекунд. Под основными действиями пользователя понимаются такие действия, как установка точек останова, управление исполнением программы, редактирование регистров и прочие действия, выполняемые на уже запущенной программе.

2.2 Структура программной реализации

Инструмент представляет из себя совокупность нескольких сервисов, общающихся между собой по различным протоколам. Свой функционал для пользователей он предоставляет по протоколу HTTP.

Разрабатываемый инструмент взаимодействует с двумя видами внешних потребителей: с пользователями (студентами, преподавателями и администраторами) и с системами управления обучением (Learning Management Systems, LMS). Для аутентификации студентов перед инструментом используется стандартный протокол OAuth 1.0[10]. При обращении к системе, браузер студента передаёт специальное сообщение, полученное от сервера LMS.

Это сообщение сформированно согласно протоколу LTI[12] и криптографически подписано в соответствии с протоколом OAuth. Результаты проверки решений студентов передаются в систему управления обучением по протоколу LTI Basic Outcomes[18]. Этот протокол поддерживает выставление оценки за задание («ресурс» в терминологии LTI) в виде десятичного дробного числа в диапазоне от $0{,}0$ до $1{,}0$ включительно.

На рисунке 1 представлена схема взаимодействия сервисов в системе, как между собой, так и с внешними потребителями.

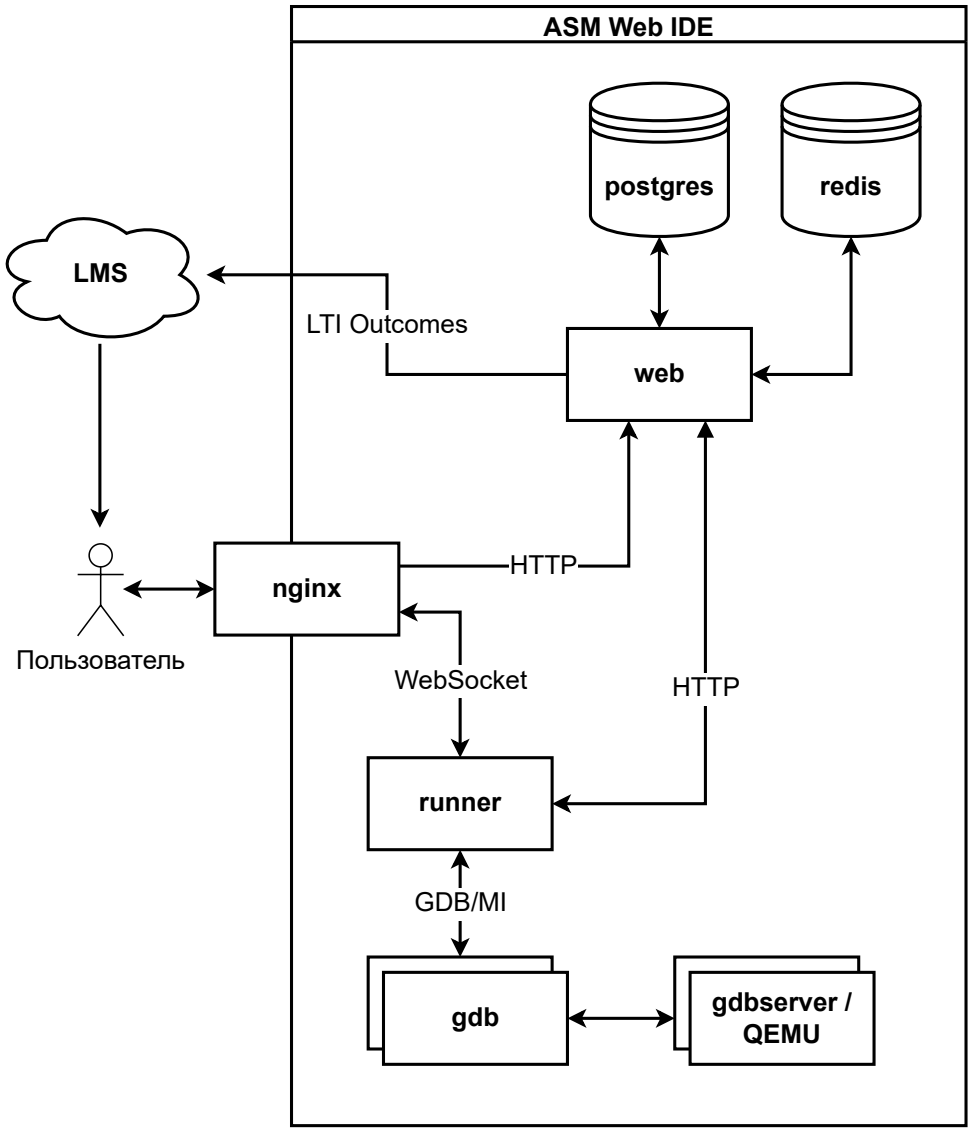

Рис. 1: Схема взаимодействия процессов в системе

Сервис **web** отвечает за основную бизнес-логику инструмента. Сервис предоставляет функционал аутентификации и авторизации, как через про-

токол LTI, так и по паре логин/пароль. Для студентов сервис предоставляет интерфейс просмотра задач и отправки решений, а для преподавателей — панель управления, позволяющую создавать и редактировать задачи, просматривать попытки решения и информацию о пользователях.

Сервис **runner** отвечает за управление сессиями отладки, взаимодействие с процессами отладчика через протокол GDB/MI, а также за взаимодействие с пользователями через протокол WebSocket. Помимо предоставления возможности интерактивной отладки веб-интерфейсу, этот сервис также занимается автоматизированной проверкой решений. Эта функциональность недоступна для внешнего пользователя напрямую, запросы на проверку решений отправляет сервис web по протоколу HTTP.

Базы данных **postgres** и **redis** используются для хранения необходимой для работы системы информации. В PostgreSQL[26] хранится информация о пользователях, задачах, заданиях и посылках. Также там хранится мета-информация об интерактивных сессиях отладки. В Redis[29] хранится множество использованных значений nonce при авторизации систем управления обучением по протоколу OAuth 1.0.

Веб-сервер **nginx** используется в качестве обратного прокси-сервера, проксирующего HTTP запросы в сервис web и взаимодействие по протоколу WebSocket с сервисом runner. Также Nginx отдаёт статические файлы, необходимые для работы с веб-сервисом, такие как сценарии на языке Javascript и файлы CSS.

2.3 Модель данных

Разрабатываемая система в своей работе оперирует различными сущностями. В этом разделе приводится описание этих сущностей и их взаимодействия между собой.

Для своей работы, система должна иметь доступ к основной информации о системах управления обучением, с которыми она взаимодействует. Эта информация представляется в виде сущности под названием tool consumer (потребитель инструмента), так как именно такое название используется в описании протокола LTI[12]. Сама система, в свою очередь, выполняет роль

tool provider (поставщик инструмента). Протокол LTI описывает, как потребители и поставщики инструментов взаимодействуют. Для взаимодействия между собой, обоим сторонам нужно знать общий ключ и секретную строку.

Система также оперирует пользователями и сохраняет определённую информацию о них. К такой информации относится, например, почтовый адрес, полное имя пользователя и метод входа в систему: через привязку к определённому потребителю инструмента, или через предоставление имени пользователя и пароля. Также, система различает привилегированных и непривилегированных пользователей. Привилегерованные пользователи имеют доступ ко всем сущностям системы, а также могут добавлять и удалять интеграции с внешними системами управления обучением.

Ещё одной сущностью, которая необходима для работы системы, является задача. Задачи привязываются к конкретному потребителю инструмента при первом запуске (LTI launch [12]). Система сохраняет различную информацию о задаче, такую как её название, её условие, а также информацию, необходимую для проверки решений по этой задаче. К такой информации относится, например, архитектура ассемблера, используемая для решений этой задачи, алгоритм проверки решений и его параметры. Для переиспользования схожих алгоритмов проверки решений между различными задачами, такие алгоритмы вынесены в отдельный класс сущностей под названием checker (проверяющая программа). Такие проверяющие программы представляют из себя подключаемые классы и используют обобщённый интерфейс для их написания.

В рамках разрабатываемой системы, такие проверяющие программы пишутся на языке Python и используют специальную библиотеку, предоставляющую доступ к многим часто используемым функциям. В свою очередь, это позволяет избавиться от необходимости формулировать задачи как набор тестов, которые подаются программе на стандартный поток ввода, а затем её вывод сравнивается с эталонным ответом. Также, такой подход позволяет выполнять проверку решений на архитектурах, не имеющих системных вызовов для вывода данных в стандартный поток вывода, например, на микроконтроллерах без какой-либо операционной системы и периферии.

Информация о связи конкретного пользователя с конкретной задачей

содержится в сущности «задание». К этой информации относится время начала выполнения задания, оценка, информация, необходимая для отправки результатов проверки потребителю инструмента через протокол LTI Basic Outcomes, а также информация о правах пользователя по отношению к этой задаче. Пользователь может быть как студентом, так и преподавателем, при этом преподаватель не обязательно является администратором всей системы. Это позволяет разграничивать доступ к различным задачам для разных пользователей. Права конкретного пользователя передаются потребителем инструмента в авторизованном сообщении LTI launch [12].

При отправке решения на проверку, система сохраняет информацию о посылке. К такой информации относится, например, исходный код, время посылки, оценка и комментарий проверяющей программы. Оценка за задание выставляется как максимум из оценок за все посылки этого пользователя по данной задаче.

Также система работает с сессиями отладки. При запуске программы в интерактивном отладчике, создаётся сессия отладки, которая привязана к конкретному пользователю и задаче. Общая информация о прошедших сессиях отладки сохраняется в базе данных. К этой информации относится запускаемый исходный код, архитектура процессора, время запуска и завершения сессии, а также объём потреблённых ресурсов: астрономического и процессорного времени, а также объём использованной оперативной памяти.

2.4 Архитектура сервиса runner

Сервис runner представляет из себя приложение, написанное на языке Python с использованием библиотек AsyncIO и AIOHTTP. Выбор данных библиотек обусловлен необходимостью асинхронного взаимодействия, как с пользователем, так и с запускаемыми программами.

2.4.1 Структура классов

Схема основных классов в сервисе представлена на рисунке 2.

Объекты этих классов отвечают за ресурсы, которыми они непосредственно управляют, а также за ресурсы вложенных классов. Для освобож-

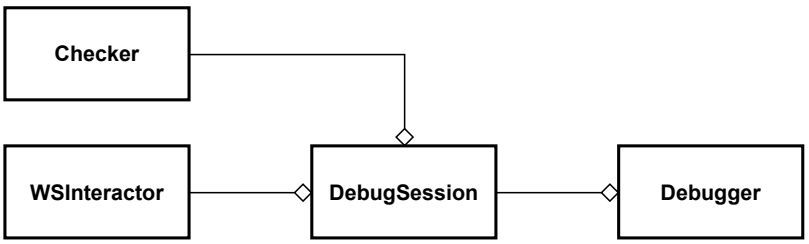

Рис. 2: Схема основных классов сервиса runner

дения ресурсов принята конвенция: каждый из этих классов предоставляет метод `close()`, освобождающий ресурсы объекта, на котором он был вызван, а также вызывающий такой же метод на вложенных объектах. Это позволяет использовать функцию `contextlib.closing` для построения контекстов оператора `with`. Управление ресурсами в программах на языке Python через использование подобных менеджеров контекстов является хорошей практикой, потому что гарантирует детерминированную сборку ресурсов, даже при наличии исключений [28].

Класс `Debugger` отвечает за взаимодействие с процессом отладчика GDB. Он абстрагирует внутри себя запуск GDB, а также отправку ему команд и получение ответов от него.

Класс `DebugSession` включает в себя как поле объект класса `Debugger`. Сам этот класс отвечает за сессию отладки: компиляцию и запуск самой отлаживаемой программы, а также предоставляет интерфейс для взаимодействия с ней, например, методы чтения и записи в регистры и память, а также методы для работы с точками останова, управления пошаговым исполнением и получении информации о использованных ресурсах. Эти методы формируют команды отладчику, исполняют их и преобразуют ответы в удобный для потребителя вид.

Класс `WSInteractor` отвечает за взаимодействие с пользователем интерактивного окна отладки по протоколу WebSocket. Протокол WebSocket позволяет передавать UTF-8 строки [6], `WSInteractor` общается с клиентом, передавая JSON объекты, сериализованные в такие строки. Такие объекты могут содержать команды от пользователя или информацию о статусе запущенной программы. Таким образом, данный класс служит адаптером для взаимодействия с сессией отладки по протоколу WebSocket.

Класс `Checker` является родительским классом для разработки проверяющих программ. При проверки решения студента, создаётся объект дочернего класса `Checker`, соответствующий нужной проверяющей программе. Сама такая программа использует интерфейс класса `DebugSession` для выполнения различных действий над программой студента. Например, проверяющая программа может записать какой-либо объект в память, выполнить функцию, а затем сверить значение в регистре проверяемой программы с нужным.

2.4.2 Компиляция ассемблерных программ

В качестве компилятора система использует GCC. GCC (GNU Compiler Collection) представляет собой набор компиляторов для различных языков программирования, в том числе и для ассемблера. GCC позволяет генерировать машинный код на самые разные архитектуры, включая нужные нам x86-64 и AVR [37]. Для архитектур x86 и x86-64 поддерживаются два диалекта ассемблера: AT&T и Intel.

При составлении задач часто бывает так, что перед проверкой решения необходимо добавлять программный код к началу или концу решения учащегося. Это вызывает несоответствие номеров строчек в коде и сообщениях об ошибках. GCC позволяет использовать в ассемблерном коде директиву `#line` для перенумерации строк.

Компиляция ассемблерных программ для архитектуры x86-64 производится с использованием следующих аргументов компилятора:

1. `-nodefaultlibs`. Данный параметр выключает связывание с библиотеками по умолчанию, необходимых в программах на языках C и C++, но не нужных для решения тех задач на языке ассемблера, для которых предназначается данная система.

2. `-nostartfiles`. Этот параметр выключает связывание с объектным файлом `crt0.o`. Так как связывание со стандартной библиотекой языка C выключено, попытка связывания с этим объектным файлом окончится неудачей.

3. `-static`. Этот параметр включает режим статического связывания. Так как программа не использует никаких разделяемых библиотек, данный флаг помогает уменьшить размер исполняемого файла и время компиляции.

4. `-g`. Данный параметр включает добавление отладочной информации в результирующий двоичный файл. Отладочная информация необходима для просмотра текущей строки в режиме интерактивной отладки.

5. `-Wl,-entry=_start_seccomp`. Данный параметр задаёт точку входа в программу вместо стандартной `_start`. Точка входа переопределяется для перевода программы в безопасный режим, как показано в исходном коде 1.

В свою очередь, компиляция ассемблерных программ для архитектуры AVR производится с использованием других аргументов компилятора. Это обусловлено тем, что в архитектуре AVR нет такого понятия, как разделяемая библиотека, а также тем, что нет необходимости запрещать доступ к определённым системным вызовам, так как весь код запускается внутри эмулятора. Параметры компилятора приведены ниже:

1. `-mmcu=atmega328p`. Данный параметр выбирает целевой микроконтроллер. В качестве такого микроконтроллера, был выбран Atmel ATmega328P из-за своей популярности, связанной с его использованием в одноплатном компьютере Arduino UNO [2].

2. `-g`. Как и в случае с x86-64, данный параметр включает добавление отладочной информации в результирующий двоичный файл.

2.4.3 Изоляция ассемблерных программ

Сервис runner использует несколько механизмов для обеспечения изоляции и ограничения потребляемых ресурсов пользовательскими программами.

Для запуска пользовательских программ используются Docker контейнеры. Docker позволяет ограничивать использование процессорного времени

и используемую память в контейнерах через механизмы контрольных групп [31]. Общее используемое процессорное время можно ограничить системным вызовом `setrlimit`, используя параметр `RLIM_CPU` [35]. Docker позволяет это делать через параметр `ulimit` в командах запуска контейнеров. Стоит заметить, что такой способ также ограничивает другие программы, запускаемые в этом же контейнере. Например, при эмуляции с помощью QEMU, данное ограничение будет применено к самому процессу QEMU. Создание и управление Docker контейнерами производится через Docker Engine API. Данное API предоставляет REST-подобный HTTP-интерфейс для управления контейнерами.

Также, Docker позволяет изолировать программы по сети, ограничивать размер дискового пространства и накладывать другие ограничения на запускаемую программу. Но даже при запуске программы с минимальными привелегиями в изолированном по сети, процессорному времени, памяти и диску контейнере, программа может совершать нежелательные действия. К таким действиям, например, можно отнести использование системного вызова `ptrace`, позволяющего определять и запрещать подключение отладчика к программе.

Поэтому, для программ архитектуры x86-64 используется только изоляция по памяти и процессорному времени. Для всех остальных нежелательных системных вызовов, сервис runner использует `seccomp` — механизм ядра Linux, позволяющий процессу перейти в «безопасный режим», в котором запрещены все системные вызовы, кроме `exit`, `sigreturn`, `read` и `write` [34]. Запретить работать со стандартными потоками ввода/вывода можно, вызвав `close` на них перед переходом в безопасный режим. Так, исходный код 1 переводит программу на языке ассемблера x86-64 в безопасный режим, предварительно закрыв стандартные потоки ввода, вывода и ошибок. Для его использования необходимо переопределить точку входа в программу, это можно сделать, передав параметр `--entry` компоновщику.

При использовании архитектуры AVR, запуск ассемблерных программ происходит с помощью эмулятора QEMU. Помимо прочего, QEMU позволяет запускать образы программ для микроконтроллеров на основе семейства архитектур AVR в режиме полносистемной эмуляции. В этом случае, процесс

```
_start_seccomp:
    mov $3, %rax        # SYS_close
    mov $0, %rdi        # STDIN_FILENO
    syscall

    mov $3, %rax        # SYS_close
    mov $1, %rdi        # STDOUT_FILENO
    syscall

    mov $3, %rax        # SYS_close
    mov $2, %rdi        # STDERR_FILENO
    syscall

    mov $157, %rax      # SYS_prctl
    mov $22, %rdi       # PR_SET_SECCOMP
    mov $1, %rsi        # SECCOMP_MODE_STRICT
    syscall

    xor %rax, %rax
    xor %rdi, %rdi
    xor %rsi, %rsi
    jmp _start
```

Исходный код 1: Перевод программы в безопасный режим

QEMU изолируется через Docker, аналогично случаю запуска кода x86-64 без использования эмулятора.

2.4.4 Отладка ассемблерных программ

Для запуска пользовательского кода в режиме отладки используется GDB. GDB (GNU Debugger) представляет из себя консольный инструмент отладки программ. GDB Позволяет отлаживать программы на самых разных языках программирования на разных платформах [36]. Также инструмент поддерживает отладочную информацию в формате DWARF, что позволяет, например, узнавать, на какой конкретно строке исходного кода находится сейчас исполнение.

При запуске программы под архитектурой x86-64, сервис runner создаёт

Docker контейнер для целей изоляции и запускает в нём утилиту под названием GDB server. Эта утилита предоставляет «мост» для GDB для отладки программы внутри запущенного контейнера. Это позволяет не ограничивать сам процесс GDB в ресурсах вместе с запускаемой программой. GDB сервер общается в GDB по протоколу TCP внутри одного физического хоста, но между разными контейнерами.

Так как GDB сервер работает только для той архитектуры процессора, на котором он запускается, при запуске ассемблерных программ с использованием эмуляторов, необходим другой механизм взаимодействия с GDB. Для этих целей, QEMU предоставляет свой собственный сервер, реализующий протокол GDB под названием GDB stub [8]. В отличие от GDB сервер, он не реализует некоторые полезные расширения протокола взаимодействия GDB Remote Serial Protocol, такие как передача файлов и перезапуск программы. Поэтому, при запуске программ на архитектуре AVR, для их перезапуска используется установка регистра PC в значение 0. Для микроконтроллера ATmega328P это соответствует вызову обработчика прерывания RESET, который вызывается при перезагрузке или включении микроконтроллера [4].

2.4.5 Взаимодействие с GDB

Для взаимодействия с процессом GDB используется GDB/MI. GDB/MI (GDB Machine Interface, машинный интерфейс GDB) — это один из форматов взаимодействия с GDB. В то время, как формат взаимодействия по умолчанию ориентирован на интерактивные терминальные сенасы, GDB/MI предназначен в первую очередь для использования другими программами, в которых отладчик является лишь одним компонентом целой системы [36]. Для запуска GDB в таком режиме используется параметр командной строки `--interpreter=mi2`.

Протокол GDB/MI использует текстовый канал связи между клиентом и процессом GDB. Сообщения в нём разделяются переводом строки. Клиент может отправлять команды, отличные от тех, которые используются в обычном режиме взаимодействия. Такие команды начинаются с символа `-`, а их аргументы разделяются пробелами. GDB поддерживает большое количество

различных команд, но разрабатываемый инструмент использует некоторые из них:

1. `-target-select`. Данная команда позволяет подключиться к GDB серверу или QEMU, запущенным внутри контейнера.

2. `-gdb-set`. Эта команда позволяет устанавливать значения различных переменных GDB. В инструменте используется для задания пути к исполняемому файлу, а также для установки значений регистров.

3. `-file-exec-and-symbols`. Эта команда аналогична команде `file` обычного режима работы с GDB. Используется для указания файла для чтения отладочной информации.

4. `-break-insert` и `-break-delete`. Эти команды используются для управления точками останова, аналогично командам `break` и `delete` консольного режима GDB.

5. `-data-list-register-names` и `-data-list-register-values`. Данные команды позволяют получать информацию о состоянии регистров программы. При запуске в интерактивном отладчике, данная информация показывается пользователю. Если же программа запускается в режиме проверки, то эту информацию проверяющая программа может использовать для определения корректности посылки.

6. `-data-read-memory-bytes` и `-data-write-memory-bytes`. Данные команды позволяют читать и записывать данные в оперативной памяти отлаживаемой программы. Могут быть использованы проверяющими программами для взаимодействия с посылками пользователя.

7. `-data-evaluate-expression`. Позволяет расчитывать произвольные выражения и используется для реализации просмотра переменных в интерактивном отладчике.

8. `-exec-interrupt` и `-exec-continue`. Эти команды позволяют приостанавливать исполнение программы и продолжать его. Используются как и в интерактивном отладчике, так и при проверке решений.

9. `-exec-next-instruction`, `-exec-step-instruction` и `-exec-finish`. Данные команды позволяют исполнять программу по шагам и используются для реализации соответствующих действий в интерактивном отладчике.

Сам процесс GDB может посылать несколько разных типов сообщений клиенту. В протоколе GDB/MI тип сообщения определяется его первым символом, а сами сообщения продолжаются до перевода строки. Сообщения могут включать в себя произвольные данные, состоящие из чисел, строк, списков и словарей. Списки и словари, в свою очередь, содержат данные в таком же формате. Для разбора сообщений был написан синтаксический анализатор, использующий метод рекурсивного спуска[9]. Разрабатываемый инструмент реагирует на следующие типы сообщений:

1. Результат выполнения команды (`result-record`). Подобные сообщения начинаются с символа `^`. Они содержат результат обработки последней клиентской команды, в частности завершилась ли она успешно или нет. Для команд, которые запрашивают какое-либо состояние отладчика или запущенной программы, данные сообщения могут также включать эту запрашиваемую информацию.

2. Асинхронная информация об исполнении (`exec-async-output`). Подобные сообщения начинаются с символа `*`. Они содержат информацию об изменении состояния запущенной программы, её переход из приостановленного в исполняющееся состояние и наоборот. Разрабатываемый инструмент использует эту информацию для отображения текущего состояния программы в интерактивном отладчике, а также для того, чтобы понимать, когда программа остановилась при её исполнении до какой-либо точки.

3. Асинхронные уведомления (`notify-async-output`). Подобные сообщения начинаются с символа `=`. Они могут содержать различную информацию об изменении текущего состояния программы, не связанной непосредственно с её остановкой или возобновлением. Разрабатывае-

мый инструмент перехватывает подобные сообщения о создании и уничтожении групп потоков. Это позволяет определить идентификатора процесса отлаживаемой программы внутри контейнера, что позволяет измерять потребление процессорного времени и памяти для конкретно этого процесса.

Информация о протоколе GDB/MI подробно описана в [36].

2.5 Интерфейс пользователя

Программная реализация инструмента предоставляет возможность взаимодействовать с ним по протоколу HTTP. В этом разделе приведено описание различных видов взаимодействия с системой. За формирование страниц инструмента отвечает сервис web. Он представляет из себя веб-приложение, написанное на языке программирования Python с использоваением библиотеки Flask. Выбор данной библиотеки обусловлен большим количеством сторонних расширений, предназначенных для решения частых задач, возникающих при разработке веб-приложений.

2.5.1 Запуск инструмента с помощью протокола LTI

Для взаимодействия с обучающими системами по протоколу LTI используется библиотека `lti`. Данная библиотека предоставляет утилиты для разбора подписанных согласно протокол OAuth сообщений в формате LTI, а также механизмы отправки таких сообщений [11].

На рисунке 3 показана диаграма взаимодействия браузера пользователя, обучающей системы и разрабатываемого инструмента при открытии студентом задания. При запросе страницы урока, система управления обучением создаёт специальное сообщение, подписанное согласно протоколу OAuth 1.0 [12]. Данное сообщение содержит большое количество различных параметров, ниже приводится описание тех, которые используются в разрабатываемом инструменте

1. `oauth_version`. Данное поле содержит версию протоколу OAuth. Согласно спецификации LTI 1.1, данное поле всегда должно иметь значе-

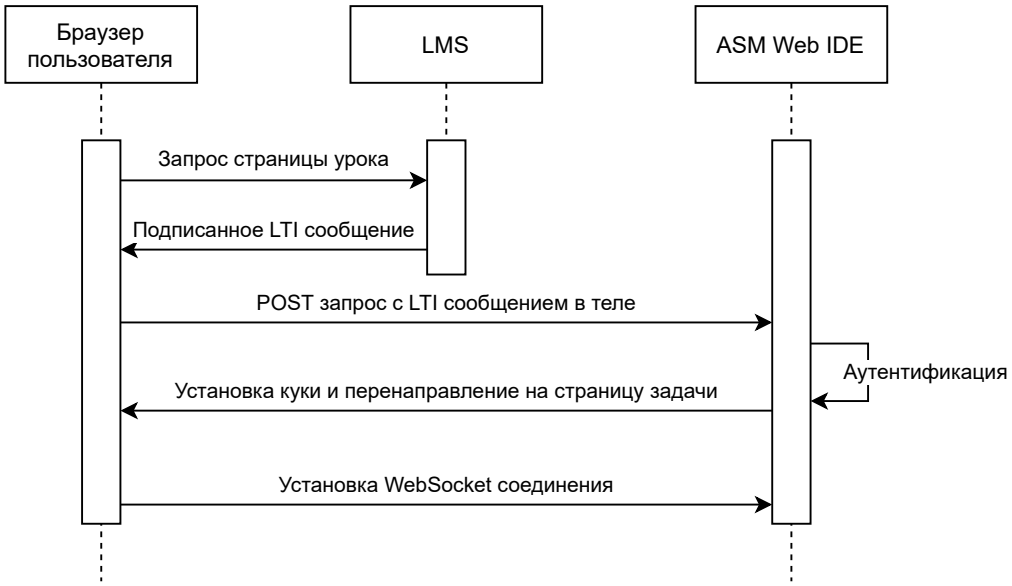

Рис. 3: Диаграма запуска инструмента через LTI интеграцию

ние `1.0`.

2. `oauth_nonce` и `oauth_timestamp`. Данные поля используются для защиты от повторных отправок одного и того же сообщения. Значения `oauth_nonce` сохраняются в Redis. В поле `oauth_timestamp` должна содержаться метка Unix-времени отправки запроса. Это время не должно отличаться от системного времени сервера больше, чем на десять минут.

3. `oauth_consumer_key`, `oauth_signature` и `oauth_signature_method`. Данные поля предоставляют возможность аутентифицировать автора LTI сообщения, как конкретную систему управления обучением, а также убедиться в целостности этого сообщения.

4. `user_id`. Данное поле содержит уникальный идентификатор пользователя в системе управления обучением. Если в базе данных инструмента ещё нет информации об этом пользователе, то она создаётся и привязывается к нужной LMS.

5. `lis_person_contact_email_primary` и `lis_person_name_full`. Данные поля содержат персональную информацию о пользователе: его адрес электронной почты и полное имя соотвественно. Согласно специ-

фикации, отсутствие этих полей означает то, что эти данные скрыты настройками приватности. Таким образом, даже если эти данные уже были сохранены в раньше, если при очередном запуске инструмента они не указаны, их требуется удалить из базы данных [12].

6. `context_title` и `resource_link_title`. Значения этих полей используются в качестве названия курса и названия задачи по умолчанию.

7. `lis_result_sourcedid` и `lis_outcome_service_url`. Данные поля определяют, по какому адресу и идентификатору отправлять информацию о оценке решения студента. Эта информация передаётся по протоколу LTI Basic Outcomes [18].

8. `roles`. Данное поле содержит разделённый запятыми список ролей пользователя относительно этой конкретной задачи. При наличии роли `Instructor`, инструмент предоставляет доступ к редактированию задачи, просмотру решений по ней и прочих привилегированных действия. При отсутствии такой роли в списке, считается что у пользователя нет никаких специальных прав касаемо этой задачи.

Таким образом, при запуске инструмента, браузер пользоватля направляет HTTP POST запрос на URL `/lti` с этим сообщением. Инструмент авторизует пользователя, создавая его сущность в базе данных, если такой пользователь не найден. Также, инструмент может обновить информацию о задаче и пользователе, если она не совпадает с той информацией, которая содержится в LTI сообщении. После этого, сервис web возвращает перенаправление на страницу задачи, содержащую в том числе и интерфейс интерактивного отладчика.

2.5.2 Взаимодействие с интерактивным отладчиком

Графический интерфейс интерактивной отладки позволяет устанавливать точки останова, просматривать и изменять значения регистров, просматривать значения произвольных выражений. Также интерфейс предоставляет

возможность приостанавливать и возобновлять исполнение программы, а также исполнять её пошагово. Рисунок 4 показывает, как выглядит интерфейс пользователя при работе с интерактивным отладчиком на примере задачи о длине строки. Данная страница генерируется сервисом web и возвращается при GET-запросе на адрес `/assignments/`*id*, где *id* — уникальный идентификатор задания в системе.

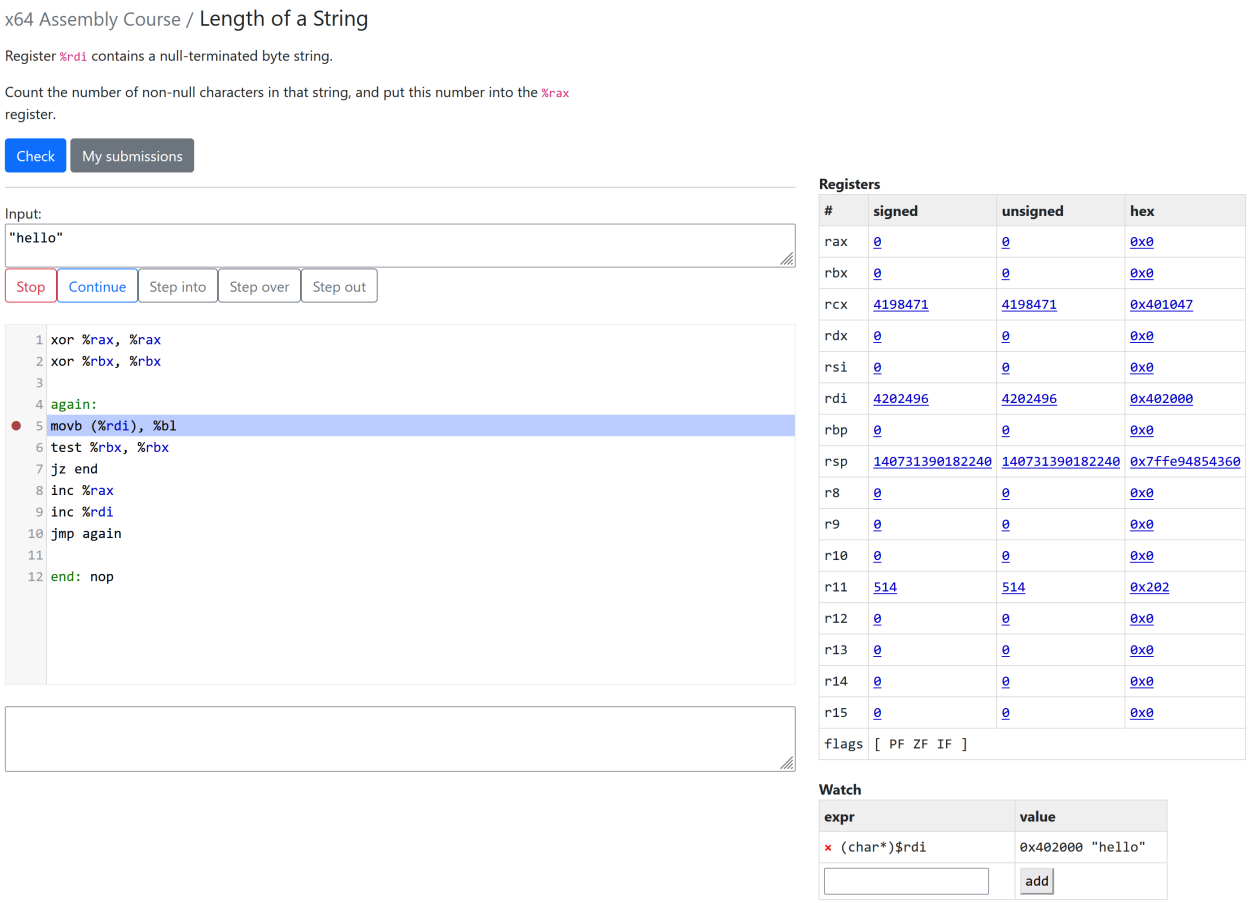

Рис. 4: Пример запуска программы в интерактивном отладчике

При отладке студентом решения, может возникнуть необходимость передать запускаемой программе определённые данные до начала работы. Так, если программа ожидает указатель на нуль-терминированную строку в определённом регистре, нужно не только установить значение этого регистра, но и записать такую строку в нужный участок памяти. Если задача сконфигурирована с проверяющей программой, поддерживающей данный механизм, то пользователю предлагается поле для ввода произвольных входных данных в

виде текста. Этот текст называется тестовыми данными. Проверяющая программа, в свою очередь, получает эти тестовые данные, и, в зависимости от их содержимого, исполняет произвольные действия с отлаживаемой программой до того, как управление над ней передастся пользователю интерактивного отладчика. К таким действиям, например, могут относиться запись в регистр или запись данных в память.

Для интерактивной отладки программ необходимо не только передавать команды из веб-интерфейса в GDB, но и асинхронно реагировать на события, возникающие при отладки. К таким событиям, например, относится остановка программы на точке останова. К счастью, все современные браузеры поддерживают протокол WebSocket[38], который позволяет общаться клиенту и серверу полностью асинхронно, а не по модели запрос-ответ.

При установке соединения по протоколу WebSocket, браузер пользователя направляет запрос в сервис web по пути `/assignment/`*id*`/websocket`, где *id* — идентификатор задания. Сервис, в свою очередь, проверяет корректность данных и прав доступа, а затем возвращает перенаправление на закрытую конечную точку через заголовок `X-Accel-Redirect`. Nginx сконфигурирован так, что, видя такой заголовок от источника, он начинает перенаправлять весь последующий трафик в сервис runner. Весь последующий процесс интерактивной отладки происходит по протоколу WebSocket напрямую между браузером пользователя и сервисом runner.

Сообщения, которыми обмениваются браузер пользователя и сервис runner через протокол WebSocket, представляют из себя JSON объекты, сериализованные в строки. Поле `type` в этих объектах определяет тип сообщения. Браузер пользователя может отправлять следующие типы сообщений сервису:

1. `run`. В качестве параметров, браузер передаёт исходный код, исходный список точек останова, исходный список отслеживаемых выражений, а также тестовые данные, на которых запускается программа.

2. `kill`. Останавливает сессию отладки и завершает исполнение программы.

3. `continue`, `step_into`, `step_over`, `step_out`. Исполняет соответствующие действия над отлаживаемой программой.

4. `pause`. Приостанавливает исполнение программы, если она исполняется в настоящий момент. Данная функция позволяет прервать программу, вошедшую в бесконечный цикл, без необходимости её полной остановки.

5. `add_breakpoint` и `remove_breakpoint`. Позволяют управлять точками останова. Данные команды принимают параметр `line`, соответствующий номеру строки в исходном файле, на которую требуется поставить или с которой требуется удалить точку останова

6. `get_registers`. Данный пакет запрашивает текущее состояние регистров программы. В ответ, сервер должен прислать пакет типа `registers`.

7. `update_register`. Данное сообщение содержит в себе название регистра и его новое значение. При получении этого сообщения, сервер изменяет значение соответствующего регистра в программе.

8. `add_watch` и `remove_watch`. Позволяют управлять наблюдаемыми выражениями. Оба принимают параметр `expr`, содержащий текстовое представление выражения, которое требуется удалить или добавить в окно просмотра.

В свою очередь, инструмент отправляет браузеру пользователя сообщения следующих типов:

1. `compilation_result`. Пакет с данным типом отправляется сервером после завершения компиляции программы. Данный пакет содержит в себе флаг успешности компиляции, а также вывод компилятора для отображения пользователю.

2. `finished`, `running` и `paused`. Эти сообщения уведомляют клиента об изменении состояния программы. Они нужны для корректного отображения пользовательского интерфейса в различных ситуациях. Напри-

мер, прервать программу можно только в том случае, когда она в данный момент работает, а не приостановлена.

3. `registers`. Данное сообщение содержит в себе состояние регистров для отображения в пользовательском интерфейсе. Регистры, отправляемые для каждой архитектуры, определяются для этих архитектур отдельно в файле конфигурации сервиса runner.

4. `watch`. Данное сообщение содержит информацию о результатах исполнения отслеживаемых выражений и необходимо для отображения этих результатов в пользовательском интерфейсе.

5. `output`. Это сообщение позволяет выводить в лог, отображаемый в браузере, произвольные данные. Используется для вывода сообщений о причинах завершении программы.

6. `error`. Подобные сообщения отправляются сервером при обнаружении какой-либо критической ошибки, например, неправильного формата запроса от клиента или неожиданного ответа GDB. После отправки такого сообщения, WebSocket соединение автоматически закрывается.

2.5.3 Интерфейс преподавателя

Сам по себе инструмент не предоставляет возможности создать новую задачу. Задачи, а также права пользователей по отношению к ним, синхронизируются с помощью протокола LTI. Поэтому, для создания новой задачи, необходимо добавить новую активность, использующую разрабатываемую систему как внешний инструмент по протоколу LTI, в системе управления обучением. Тогда, при первом запросе к странице активности, будет создана соответствующая задача с заголовком и названием курса, соответствующим заданным в настройках потребителя инструмента. Информация о правах доступа к задаче также будет синхронизирована. Если у пользователя есть доступ инструктора в системе управления обучением, то тогда, у него будет доступ к редактированию настроек задачи, а также к странице просмотра решений других пользователей.

Страница редактирования задачи имеет URL `/admin/problem/`*id*, где *id* — идентификатор задачи. При GET запросе к этой странице сервис web проверяет доступы текущего пользователя, а затем возвращает форму редактирования задачи. При POST запросе, происходящим при отправке формы, её данные проверяются на корректность, проверяется, что у пользователя достаточно прав для редактирования этой задачи, а затем изменения сохраняются в базе данных.

Преподаватель может просмотреть список посылок пользователей по адресу `/admin/problem/`*id*`/submissions`. На этой странице можно просмотреть список пользователей, когда-либо открывавших задачу, а также статус её выполнения ими. Для каждого пользователя доступен список посылок, вместе с исходными кодами и датами выполнения.

В разрабатываемом инструменте некоторые пользователи могут иметь статус администратора. В таком случае, они автоматически получают доступ к управлению всеми задачам и пользователями. Также, только администраторы могут добавлять интеграции с системами управления обучением. Пользователя с правами администратора можно создать, выполнив команду `./manage.py flask create-admin`.

2.6 Контроль состояния инструмента

Для контроля текущего состояния инструмента используется Prometheus [27]. Prometheus — это программное обеспечение с открытым исходным кодом, собирающее различные сигналы и аггрегирующее их для построения графиков и предупреждений. Разрабатываемый инструмент отсылает некоторую числовую информацию в Prometheus, чтобы администратор системы мог просматривать и анализировать её. Основную информацию о состоянии инструмента предоставляет сервис runner. К этой информации относятся:

1. Количество одновременно работающих сессий отладки. Этот параметр показывает общую нагруженность системы, ведь каждая сессия потребляет оперативную память и другие ресурсы сервера на всём своём протяжении. Сервис runner отдаёт этот сигнал с информацией об архитектуре, что позволяет анализировать, сколько именно сессий отладки

запущено с использованием разных архитектур.

2. Статистика о количестве и успешности исполнения команд GDB. Этот сигнал позволяет измерять, сколько команд отправляется отладчикам GDB в секунду, а также сигнализировать при их неуспешном выполнении. Данный сигнал содержит в себе информацию о типе команды.

3. Гистограмма времени обработки команд GDB. Тип сигнала «гистограмма» позволяет принимать измерения, чьё распределение заранее неизвестно. Такие сигналы можно анализировать разными способами, такими как построение среднего или персентиля за какой-либо интервал. Данный сигнал разбит на группы по каждому типу команд, так как время исполнения различных команд может значительно отличаться.

4. Гистограмма времени компиляции и запуска программ. Этот сигнал позволяет судить, сколько времени проходит от запуска программы в интерактивном отладчике до возможности с ней взаимодействовать. Сигнал сгруппирован по архитектуре, на которой запускается программа.

2.7 Запуск и развёртывание системы

Разрабатываемый инструмент состоит из нескольких сервисов, работающих параллельно и взаимодействующих друг с другом, что усложняет запуск системы. Для удобства разработки и развёртывания, был написан специальный сценарий `manage.py`, который позволяет одной командой запустить все нужные сервисы. Этот сценарий использует Docker Compose[24] для автоматического развёртывания нескольких Docker контейнеров, исходя из их описания. Сценарий предоставляет возможность для перезапуска конкретных сервисов в процессе работы системы. Также он предоставляет простой способ запустить команду внутри контейнера, что может быть необходимо для обслуживания сервиса.

Команда `./manage.py run` позволяет запустить инструмент. При этом постоянные данные будут сохраняться в Docker томах. Данный режим работы сценария принимает различные параметры. Так, параметр `--port` указывает,

к какому TCP порту привязать запущенный HTTP сервер. Параметр `--detach` позволяет запустить сервис в фоновом режиме, а параметр `--prod` указывает на необходимость отключения различных отладочных функций.

Чтобы перезапустить один из работающих сервисов, необходимо выполнить команду `./manage.py restart <имя сервиса>`. Все процессы инструмента можно остановить командой `./manage.py stop`.

Командой `./manage.py shell` можно открыть интерпретатор командной строки внутри контейнера с сервисом web, а команда `./manage.py flask` позволяет исполнять различные действия в контексте Flask-приложения. Так, например, команда `./manage.py flask db upgrade` приводит состояние базы данных в самое новое, применяя все неприменённые миграции.

Глава 3. Исследование свойств решения

3.1 Измерение потребляемой памяти и процессорного времени

Запуск ассемблерной программы как в режиме интерактивной отладки, так и в режиме проверки решения неизбежно расходует серверные ресурсы. Для того, чтобы оценить количество пользователей, способных одновременно запускать свои программы в разрабатываемом инструменте, необходимо измерить, сколько процессорного времени и оперативной памяти используют сессии отладки.

Каждая сессия отладки представляет из себя несколько процессов. В случае архитектуры x86-64 к таким процессам относятся GDB, GDB сервер и сама запущенная программа. При этом, GDB сервер и программа запускаются в отдельном Docker контейнере в целях изоляции. В случае использования архитектуры AVR, вместо GDB сервера, который запускает программу, используется один процесс эмулятора QEMU. Также, запущенные сессии отладки могут увеличивать потребление ресурсов самим сервисом runner.

Так как Docker контейнеры используют механизм контрольных групп для изоляции, информацию о потреблении памяти и процессорного времени такой группой можно прочитать из различных файлов в каталоге `/sys/fs/cgroup`. Так, информация о количестве потраченных наносекунд прроцессорного времени, содержится в файле `/sys/fs/cgroup/cpuacct/cpuacct.usage`, а из файла `/sys/fs/cgroup/memory/memory.usage_in_bytes` можно прочитать количество байт используемой оперативной памяти.

Для анализа потребления ресурсов используется инструмент cAdvisor[25], позволяющий передавать эти и многие другие сигналы в систему Prometheus. Он также основывается на данных, предоставляемых механизмом контрольных групп. Для измерения используемой памяти был использован сигнал `container_memory_working_set_bytes`. Данный сигнал отдаётся процессом cAdvisor и представляет из себя размер занимаемой оперативной памяти, которую система не может освободить для других процессов. В это число не входят, например, кешируемые операционной системой файлы. Для измерения используемого процессорного времени используется сигнал

`container_cpu_usage_seconds_total`. Для анализа и аггрегации полученных сигналов была использована система Grafana[16].

3.2 Нагрузочное тестирование

Для оценки потребления системных ресурсов инструментом, была реализована небольшая программа на языке Python, которая использует метод нагрузочного тестирования. Эта программа взаимодействует с инструментом по протоколу WebSocket аналогично тому, как бы с ним взаимодействовал интерфейс интерактивного отладчика. Для выполнения нагрузочного тестирования, с небольшой задержкой последовательно запускаются несколько программ в режиме интерактивной отладки, которые затем выполняют некоторые действия. Это позволяет выяснить количество оперативной памяти и процессорного времени, используемого системой при одновременном исполнении фиксированного количества сессий отладки. Ниже приводится описание программ и действий, используемых для нагрузочного тестирования.

1. `SingleStep`. Данный профиль запускает программу, представляющую из себя пустой бесконечный цикл. Далее, в цикле, с задержкой, равномерно распределённой на интервале от 0 до 1 секунд, выполняется одно из трёх действий: либо переход на следующую инструкцию, либо установка случайного регистра в случайное значение, либо добавление точки останова на начало цикла, продолжение исполнения программы до этой точки, а затем удаление этой точки останова. При изменении регистра, он выбирается так, чтобы не помешать штатному выполнению программы. Это позволяет имитировать типичный сценарий использования интерактивного отладчика, в котором исполнение пользовательской программы самой по себе занимает очень малую часть процессорного времени, поскольку почти всё время программа проводит в приостановленном состоянии.

2. `BusyLoop`. Данный профиль также запускает программу, представляющую из себя пустой бесконечный цикл. Далее, в цикле, с задержкой, равномерно распределённой на интервале от 0 до 1 секунд, выполняется продолжение исполнения программы, а затем её приостановка. Это

позволяет создать нагрузку на процессор сервера, подобную той, которая возникла бы при длительном исполнении пользовательского кода, не прерываемого отладчиком. Время выполнения программы до её приостановки также распределено случайно на интервале от 0 до 1 секунд.

Измерения были выполнены на компьютере с процессором AMD Ryzen 7 5800H, 32 гигабайтами оперативной памяти и операционной системой Arch Linux. Измерения потребляемой памяти и процессорного времени включают в себя все процессы инструмента. Это серисы web и runner, базы данных PostgreSQL и Redis и веб-сервер Nginx. Система была настроена так, чтобы контейнер с программой и сервером GDB не мог потреблять более $0{,}1$ ядра процессорного времени и 32 МиБ памяти.

Результаты измерения использования памяти представлены в таблице 2. В каждом эксперименте было запущено N одновременных сессий отладки с различными профилями, которые длились две минуты. Было произведено измерение среднего значения μ_{Mtotal} и стандартного отклонения σ_{Mtotal} общего потребления памяти инструментом за это время. Также, отдельно было произведено измерение среднего значения μ_{Mprog} и стандартного отклонения σ_{Mprog} суммарного потребления памяти контейнерами с запущенными программами и серверами GDB, без учёта остальных сервисов.

Стандартное отклонение измерений оказалось достаточно низким. Также, потребление памяти между профилями `SingleStep` и `BusyLoop` на одной и той же архитектуре при одном и том же N, оказалось схожим. При этом, потребление памяти как всей системой, так и суммарное потребление памяти контейнерами, используемыми для изоляции запускаемых программ, растёт линейно с увеличением числа одновременных сессий отладки. Формулы 1 и 2 позволяют вычислить среднее дополнительное потребление памяти μ_{Msession} одной сессией отладки, а также её стандартное отклонение $\sigma_{\mathrm{Msession}}$.

$$\mu_{\mathrm{Msession}} = \frac{1}{k} \sum_{i=1}^{k} \frac{\mu^{(i)}_{\mathrm{Mtotal}} - \mu^{(N=0)}_{\mathrm{Mtotal}}}{N_i} \tag{1}$$

$$\sigma^2_{\mathrm{Msession}} = \frac{1}{k^2} \sum_{i=1}^{k} \left(\frac{\sigma^{(i)}_{\mathrm{Mtotal}}}{N_i} \right)^2 \tag{2}$$

Профиль	N	μ_{Mtotal} (МиБ)	σ_{Mtotal} (МиБ)	μ_{Mprog} (МиБ)	σ_{Mprog} (МиБ)
Без нагрузки	0	132,22	0,07	0,00	0,00
SingleStep (x86-64)	10	277,74	0,44	10,45	0,16
SingleStep (x86-64)	50	854,40	0,34	50,74	0,30
SingleStep (x86-64)	100	1581,16	0,30	100,59	0,28
BusyLoop (x86-64)	10	275,33	0,25	10,45	0,19
BusyLoop (x86-64)	50	857,80	0,32	50,81	0,27
BusyLoop (x86-64)	100	1591,60	0,39	100,79	0,33
SingleStep (AVR)	10	270,90	0,11	77,98	0,00
SingleStep (AVR)	50	839,80	0,29	401,78	0,00
SingleStep (AVR)	100	1648,64	0,16	805,39	0,00
BusyLoop (AVR)	10	286,22	0,14	77,94	0,00
BusyLoop (AVR)	50	876,43	0,18	391,76	0,00
BusyLoop (AVR)	100	1611,98	0,20	775,42	0,00

Таблица 2: Потребление памяти инструментом в различных ситуациях

Из этих формул следует, что для архитектуры x86-64 среднее потребление памяти одной сессией отладки составляет $14{,}48$ МиБ с незначительным средним отклонением в $0{,}014$ МиБ. При этом, лишь $1{,}02$ МиБ занимают в памяти процесс отлаживаемой программы и процесс GDB сервера вместе взятые. Большая часть расходов памяти ложится на процесс GDB, запущенный в том же контейнере, что и сервис runner.

Для архитектуры AVR ситуация немного отличается. Среднее потребление памяти схоже, оно составляет $14{,}71$ МиБ, среднее отклонение измерений составляет всего $0{,}0086$ МиБ. Но в отличие от архитектуры x86-64, изолируемый контейнер с эмулятором занимает в среднем $7{,}87$ МиБ. Это компенсируется тем, что процесс `avr-gdb` потребляет меньше памяти, чем аналогичный процесс для архитектуры x86-64.

Результаты измерения использования процессорного времени представлены в таблице 3. Как и в случае с измерением памяти, в каждом эксперименте было запущено N одновременных сессий отладки с различными профилями, которые длились две минуты. Было произведено измерение среднего значения

μ_{Ptotal} и стандартного отклонения σ_{Ptotal} использования инструментом процессорного времени. Эти значения измеряются в секундах, значение μ_{Ptotal}, равное одной секунде означает, что нагрузка всех компонентов инструмента на процессор была эквивалентна полной нагрузке на одно его ядро. Также было произведено измерение этих значений только для контейнеров с запущенными программами и серверами GDB. Среднее суммарное использование процессорного времени этими контейнерами имеет значение μ_{Pprog}, стандартное отклонение — σ_{Pprog}.

Профиль	N	μ_{Ptotal} (с.)	σ_{Ptotal} (с.)	μ_{Pprog} (с.)	σ_{Pprog} (с.)
Без нагрузки	0	0,0014	0,0001	0,0000	0,0000
SingleStep (x86-64)	10	0,0574	0,0037	0,0096	0,0005
SingleStep (x86-64)	50	0,2690	0,0092	0,0469	0,0013
SingleStep (x86-64)	100	0,5060	0,0180	0,0898	0,0025
BusyLoop (x86-64)	10	0,6250	0,0197	0,5890	0,0194
BusyLoop (x86-64)	50	3,1392	0,0689	3,0125	0,0676
BusyLoop (x86-64)	100	6,1947	0,1173	5,9465	0,1139
SingleStep (AVR)	10	0,0681	0,0034	0,0150	0,0009
SingleStep (AVR)	50	0,3062	0,0126	0,0729	0,0020
SingleStep (AVR)	100	0,5784	0,0192	0,1388	0,0040
BusyLoop (AVR)	10	0,6381	0,0268	0,6023	0,0266
BusyLoop (AVR)	50	3,1596	0,0614	3,0254	0,0602
BusyLoop (AVR)	100	6,1798	0,1352	5,9399	0,1267

Таблица 3: Потребление процессорного времени инструментом в различных ситуациях

Используя формулы, аналогичные 1 и 2, можно оценить использование процессорного времени одной сессией отладки. При запуске программ профиля `SingleStep`, среднее потраченное процессорное время в пересчёте на одну сессию отладки контейнерами с программой и GDB сервером составляет $0{,}93$ мс и $1{,}45$ мс для архитектур x86-64 и AVR соответственно. Так как пользовательские программы исполняются в таких контейнерах, то можно отдельно оценить дополнительные затраты процессорного времени остальными компонентами системы без учёта пользовательских программ. Это измерение

будет более информативно, так как количество процессорного времени, которое может потратить пользовательская программа, зависит лишь от настроек системы.

При подсчёте получается, что одна сессия отладки x86-64 тратит $4{,}61$ мс процессорного времени, а одна сессия отладки AVR тратит $5{,}15$ мс. Эти значения не включают в себя траты самой запускаемой программы в случае, если она исполняется долгое время без приостановки, но включает затраты GDB, сервера GDB, QEMU и сервиса runner.

3.3 Измерение задержки интерактивного отладчика

При отладке программ с помощью веб-инструмента неизбежно возникает задержка между выполнением пользователем действия в интерфейсе, обработкой этого действия и отображения результатов пользователю. Для определения скорости реакции инструмента на действия пользователя, были исследованы задержки двух типов таких действий: самого запуска программы и элементарных действий с запущенной программой.

Для пользователя интерактивного отладчика, задержка запуска программы складывается из нескольких факторов. К ним относятся: время компиляции программы, время создания и запуска Docker контейнера, время подключения к GDB серверу и выполнения команд отладчика для настройки окружения. Для измерения этих задержек, исходный код сервиса runner был модифицирован так, чтобы он сохранял в файл информацию о времени исполнения соответствующих участков кода. Для измерения этого времени используется функция `time.monotonic()`. На ОС Linux, на которой производились измерения, эта функция использует системный вызов `clock_gettime()`. Используя системный вызов `clock_getres()`, возможно узнать разрешающую способность этого таймера (см. исходный код 2). Для компьютера, на котором запускался инструмент, эта разрешающая способность составляет одну наносекунду.

Стоит заметить, что на задержку может также влиять состояние сети, по которой происходит взаимодействие с инструментом. Экспериментальные измерения производились с клиентом и сервером на одном физическом

```c
#include <time.h>
#include <stdio.h>

int main() {
    struct timespec ts;
    if (clock_getres(CLOCK_MONOTONIC, &ts)) {
        perror("clock_getres");
        return 1;
    }
    printf("tv_sec = %ld, tv_nsec = %ld\n",
           ts.tv_sec, ts.tv_nsec);
    return 0;
}
```

Исходный код 2: Запрос разрешающей способности таймера

компьютере, поэтому эта задержка в экспериментах минимальна. Также, при выполнении измерений, инструмент не был загружен другими запросами и сессиями отладки. Всего было проведено $N = 100$ запусков программы на архитектурах x86-64 и AVR, результаты представлены в таблицах 4 и 5. Для анализа результатов были использованы библиотеки NumPy[22] и Pandas[21].

Фактор	Среднее время (с.)	Станд. отклонение (с.)
Компиляция программы	0,017	0,002
Запуск процесса GDB	0,031	0,005
Создание контейнера	0,027	0,003
Запуск контейнера	0,971	0,083
Подключение к GDB серверу	0,048	0,054
Настройка точек останова	0,001	0,000
Старт программы	0,040	0,041
Прочее	0,007	0,018
Итого	1,143	0,090

Таблица 4: Факторы задержки запуска программы (x86-64)

Как можно видеть из результатов измерений, подавляющее большинство времени (в среднем около 85%) при запуске программы тратится именно

Фактор	Среднее время (с.)	Станд. отклонение (с.)
Компиляция программы	0,010	0,002
Запуск процесса GDB	0,008	0,001
Создание контейнера	0,027	0,002
Запуск контейнера	0,946	0,069
Подключение к QEMU	0,203	0,000
Настройка точек останова	0,001	0,000
Старт программы	0,001	0,000
Прочее	0,004	0,000
Итого	1,200	0,069

Таблица 5: Факторы задержки запуска программы (AVR)

на запуск Docker контейнера с GDB сервером. Это связано с тем, что при запуске контейнера, Docker должен настроить большое количество механизмов операционной системы, а также обеспечить сетевую доступность контейнера по его имени. Это важно сделать до попытки подключения к GDB серверу, потому что при ошибке разрешения доменного имени, GDB не будет повторять попытки подключения, а вернёт ошибку.

Для измерения времени задержки при обычных действиях отладки, была запущена программа `SingleStep` на архитектурах x86-64 и AVR, которая производила свои обычные случайные действия. Всего было произведено $N = 100$ действий каждого типа. Время измерялось также используя функцию `time.monotonic()`. Задержка для команды «выполнить следующую инструкцию» и для команды «продолжить исполнение» измерялась от отправки этой команды до получения уведомления о приостановке исполнения программы. Для команды «изменить регистр», время считалось до получения обновлённого списка значений регистров. Результаты измерений представлены в таблице 6. Как можно видеть, задержка выполнения этих действий составляет 2–3 миллисекунды, что, при отсутствии значительной сетевой задержки, должно восприниматься пользователем инструмента как мгновенная реакция.

Действие	Среднее время (с.)	Станд. отклонение (с.)
Сделать один шаг (x86-64)	0,0026	0,0002
Изменить регистр (x86-64)	0,0030	0,0003
Продолжить исполнение (x86-64)	0,0034	0,0003
Сделать один шаг (AVR)	0,0026	0,0003
Изменить регистр (AVR)	0,0034	0,0003
Продолжить исполнение (AVR)	0,0036	0,0003

Таблица 6: Задержка выполнения действий

3.4 Выводы

Характеристика потребления оперативной памяти и процессорного времени разработанным инструментом позволяет поддержать более 100 типичных одновременных сессий отладки на сервере с двумя гигабайтами оперативной памяти и одним процессорным ядром. Для поддержки большего количества сессий отладки может потребоваться больше ресурсов. Для архитектуры x86-64, Основным потребителем оперативной памяти для каждой конкретной сессии отладки является процесс GDB. При выполнении программ для архитектуры AVR, используемая память разделена приблизительно поровну между процессом GDB и эмулятором QEMU. Таким образом, для дальнейших оптимизаций использования ресурсов инструментом, в первую очередь необходимо изучить способы сократить потребление памяти процессом GDB.

Задержка при запуске ассемблерных программ в интерактивном отладчике составляет в среднем 1,1–1,2 секунды без учёта задержки сети. Этот показатель схож с задержкой, обеспечиваемой сервисом OnlineGDB[23], который использует аналогичный механизм взаимодействия с процессом GDB, запущеным на сервере. Для уменьшения этой задержки, необходимо уменьшить время запуска контейнера с GDB сервером. Этого можно добиться, например, поддерживая некоторое количество простаивающих контейнеров. Например, в этих контейнерах может быть запущена команда `sleep infinity`, которая отдаёт управление планировщику операционной системы сразу после запуска и в дальнейшем не возобновляется. Затем, для запуска GDB сервера в

таком контейнере, достаточно будет выполнить команду `docker exec`, или аналогичное действие, используя Docker Engine API. Выполнение команды в уже запущенном контейнере происходит быстрее, чем запуск нового контейнера. Такой способ позволит потратить основное время на запуск контейнера задолго до того, как он понадобится.

Задержка выполнения элементарных операций отладки составляет всего несколько миллисекунд, а значит определяется лишь условиями сетевого подключения. Это означает, что при методе решения, в котором процесс отладчика запускается на удалённом сервере, эту задержку существенно снизить нельзя.

Заключение

Целью и основным результатом данной работы является реализация обучающего веб-инструмента удалённой сборки и интерактивной отладки программ на языке ассемблера. В частности, в ходе выполнения данной работы были получены следующие результаты:

1. Было проведено исследование существующих решений для запуска и отладки программ на языке ассемблера, обучения этому языку. В ходе исследования были выявлены достоинства и недостатки существующих решений.

2. Были сформированы функцицональные и нефункциональные требования к разрабатываемому инструменту, учитывающие недостатки существующих решений. К этим недостаткам были отнесены: отсутствие поддержки архитектур, не принадлежащим к семейству x86; отсутствие поддержки интерактивной отладки; отсутствие интеграции с существующими системами управления обучением; необходимость установки на компьютер пользователя; закрытый исходный код некоторых решений.

3. Был реализован инструмент обучения языку ассемблера, поддерживающий интерактивный запуск кода на архитектурах x86-64 и AVR, а также имеющий поддержку задач и интеграцию с системами управления обучением по протоколу LTI. Взаимодействие с инструментом производится через веб-интерфейс.

4. Было произведено измерение потребления ресурсов и производительности разработанного инструмента. Исследование показало, что основным ограничением количества одновременно запускаемых сессий отладки является объём доступной оперативной памяти. С малой степенью ошибки было определено количество оперативной памяти и процессорного времени, потребляемое интерактивными сессиями отладки. Каждая сессия отладки потребляет 14–15 МиБ памяти и 4–6 мс процессорного времени в секунду. Также было проведено измерение времени реакции интерактивного отладчика на действия пользователя.

Время запуска программы составило 1,1–1,2 секунды, время реакции на остальные действия — 2–3 мс.

Таким образом, цель данной работы была достигнута в полном объёме. В процессе исследования было установлено, что при запуске на архитектуре x86-64, процесс отладчика GDB является основным потребителем оперативной памяти. Следовательно, для уменьшения использования памяти инструментом, имеет смысл исследовать способы уменьшить это использование памяти GDB. Для уменьшения времени реакции на запуск программы пользователем, стоит заранее создать какое-то количество простаивающих контейнеров, а процессы GDB сервера или эмулятора QEMU запускать уже в них.

Список литературы

[1] *About Moodle — MoodleDocs*. URL: https://docs.moodle.org/400/en/About_Moodle. (дата обращения 01.05.2022).

[2] *Arduino UNO R3 Product Reference Manual*. Arduino S.r.l. Июнь 2021.

[3] *Assembly Debugger Online*. URL: http://asmdebugger.com. (дата обращения 30.04.2022).

[4] *ATmega328P Datasheet*. Atmel Corporation. 2015.

[5] Jakub Beránek. *Assembly debugger (x86)*. URL: https://kobzol.github.io/davis. (дата обращения 30.04.2022).

[6] Ian Fette и Alexey Melnikov. *The WebSocket Protocol*. Tex. отч. Internet Engineering Task Force (IETF), дек. 2011.

[7] Robert Fischer и Michael Lutonsky. *Yasp*. URL: https://yasp.me/. (дата обращения 05.05.2022).

[8] *GDB usage — QEMU 7.0.50 documentation*. URL: https://qemu.readthedocs.io/en/latest/system/gdb.html. (дата обращения 02.05.2022).

[9] Dick Grune и Ceriel JH Jacobs. *Parsing techniques: a practical guide*. Springer Science & Business Media, 2007.

[10] Eran Hammer-Lahav. *The OAuth 1.0 Protocol*. Tex. отч. Internet Engineering Task Force (IETF), апр. 2010.

[11] Ryan Hiebert. *lti — PyPI*. URL: https://pypi.org/project/lti/. (дата обращения 02.05.2022).

[12] *IMS Global Learning Tools Interoperability Implementation Guide*. Tex. отч. IMS Global Learning Consortium Inc., март 2012.

[13] GitHub Inc. *GitHub Classroom*. URL: https://classroom.github.com/. (дата обращения 30.04.2022).

[14] Katherine Kampf. *Seamless teaching and learning through GitHub Classroom and Visual Studio Code*. URL: https://github.blog/2021-08-12-teaching-learning-github-classroom-visual-studio-code/. (дата обращения 30.04.2022).

[15] Aidan Khoury. *NASM Assembly Language — IntelliJ IDEs Plugin*. URL: https://plugins.jetbrains.com/plugin/9759-nasm-assembly-language. (дата обращения 30.04.2022).

[16] Grafana Labs. *Grafana*. URL: https://grafana.com/. (дата обращения 14.05.2022).

[17] Sphere Research Labs. *Ideone — Online Compiler and IDE*. URL: https://ideone.com. (дата обращения 30.04.2022).

[18] *Learning Tools Interoperability Basic Outcomes*. Тех. отч. IMS Global Learning Consortium Inc., май 2019.

[19] One Compiler Pvt. Ltd. *OneCompiler — Write, run and share Assembly code online*. URL: https://onecompiler.com/assembly. (дата обращения 30.04.2022).

[20] Dmitriy Manushin. *SASM — simple crossplatform IDE for NASM, MASM, GAS, FASM assembly languages*. URL: https://dman95.github.io/SASM/index.html. (дата обращения 30.04.2022).

[21] Wes McKinney и др. “pandas: a foundational Python library for data analysis and statistics”. В: *Python for high performance and scientific computing* 14.9 (2011), с. 1—9.

[22] *NumPy*. URL: https://numpy.org/. (дата обращения 24.05.2022).

[23] *Online GCC Assembler — online editor*. URL: https://www.onlinegdb.com/online_gcc_assembler. (дата обращения 30.04.2022).

[24] *Overview of Docker Compose*. URL: https://docs.docker.com/compose/. (дата обращения 12.05.2022).

[25] David Porter, Dawn Chen и др. *cAdvisor*. URL: https://github.com/google/cadvisor. (дата обращения 14.05.2022).

[26] *PostgreSQL*. URL: https://www.postgresql.org/. (дата обращения 24.05.2022).

[27] *Prometheus — Monitoring system and time series database*. URL: https://prometheus.io/. (дата обращения 12.05.2022).

[28] Luciano Ramalho. *Fluent Python: Clear, concise, and effective programming*. O'Reilly Media, Inc., 2015.

[29] *Redis*. URL: https://redis.io/. (дата обращения 24.05.2022).

[30] Juan Carlos Rodríguez-del-Pino. *VPL — Virtual Programming Lab — About*. URL: https://vpl.dis.ulpgc.es/index.php/about. (дата обращения 01.05.2022).

[31] *Runtime options with Memory, CPUs, and GPUs // Docker Documentation*. URL: https://docs.docker.com/config/containers/resource_constraints/. (дата обращения 02.05.2022).

[32] JetBrains s.r.o. *CLion: A Cross-Platform IDE for C and C++ by JetBrains*. URL: https://www.jetbrains.com/clion/. (дата обращения 30.04.2022).

[33] JetBrains s.r.o. *EduTools — IntelliJ IDEs Plugin*. URL: https://plugins.jetbrains.com/plugin/10081-edutools. (дата обращения 30.04.2022).

[34] *seccomp(2) Linux Programmer's Manual*. 5.13. Авг. 2021.

[35] *setrlimit(2) Linux Programmer's Manual*. 5.13. Март 2021.

[36] Richard Stallman, Roland Pesch, Stan Shebs и др. *Debugging with GDB*. Free Software Foundation, 2022.

[37] Richard M Stallman и др. *Using the GNU Compiler Collection*. Free Software Foundation, 2022.

[38] *WebSocket Browser Compatibility*. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSocket#browser_compatibility. (дата обращения 05.05.2022).

[39] *Общая информация о Stepik — Справочный центр Stepik*. URL: https://support.stepik.org/hc/ru/articles/360000172234. (дата обращения 01.05.2022).