Содержание

Введение

В настоящее время обучение языку ассемблера является важной составляющей многих программистских курсов.

Очень часто студенты, изучающие язык ассемблера, сталкиваются с проблемами при настройки среды разработки, при использовании инструментов компиляции и отладки.

Преподаватели таких курсов также сталкиваются с проблемами организации учебного процесса.

Создание удобного, интерактивного и производительного программного инструмента удалённой сборки и отладки програм на ассемблере, представляет собой актуальную задачу.

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. Исследование возможности создания инструмента.
4. Разработка программной архитектуры инструмента.
5. Реализация инструмента.
6. Исследование свойств решения.

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

Предметом исследования является интерактивность и удобство использования таких систем.

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

Глава 1. Обзор предметной области

1.1 Критерии сравнения

Смотрим на следующие критерии:

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.

1.2 Существующие решения

1.2.1 Ideone

Ideone[1] является онлайн компилятором и средой разработки, поддерживающей более 60 языков программирования, в том числе несколько диалектов ассемблера.

Поддерживается запуск ассемблерного кода на архитектурах x86 (NASM и GNU диалекты) и x86-64 (только NASM диалект). Отладка не поддерживается.

[1] `https://ideone.com`

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[2] — это примерно то же самое, что и Ideone, правда поддерживает только x86 с NASM диалектом. Не уверен, нужно ли включать в список аналогов, или же хватит Ideone.

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 NASM 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[4], соответственно есть возможность установить его на выделенный сервер.

[2] `https://onecompiler.com/assembly`

[3] `http://asmdebugger.com`

[4] `https://github.com/dinoqqq/asmdebugger`

1.2.4 SASM (SimpleASM)

SASM[5] представляет из себя кроссплатформенную среду разработки на языке ассемблера для архитектур x86 и x86-64 с использованием диалектов NASM, GNU, FASM, MASM.

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[6].

1.2.5 JetBrains Clion + EduTools

Clion[7] — 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[8].

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

Плагин EduTools[9] 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.

---

[5] https://dman95.github.io/SASM/index.html

[6] https://github.com/Dman95/SASM

[7] https://www.jetbrains.com/clion/

[8] https://plugins.jetbrains.com/plugin/9759-nasm-assembly-language

[9] https://plugins.jetbrains.com/plugin/10081-edutools

1.2.6 GitHub Classroom + Visual Studio Code

GitHub Classroom — это сервис, позволяющий давать учебные задания в виде git-репозиториев. GitHub Classroom позволяет добавить кнопку «открыть в Visual Studio Code», 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 скрипта для автоматической проверки. В первом случае входные данные передаются программе через стандартный поток ввода, а вывод программы сравнивается с эталонным ответом.

Необходима установка Visual Studio Code, компилятора и отладчика.

GitHub Classroom имеет закрытый исходный код, установить свою копию на выделенный сервер не представляется возможным.

1.2.7 Stepik

В системе управления обучением Stepik есть режим задания Code Challenge, который позволяет проверять код, написанный на различных языках программирования.

Поддерживается NASM диалект x86 и x86-64 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. Система имеет закрытый исходный код.

1.2.8 Moodle + Virtual Programming Lab

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

Поддерживается NASM 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.

1.2.9 Git репозиторий с заданиями и скриптами для проверки

Сюда тоже надо что-то написать, но я пока не понимаю как это лучше оформить.

1.3 Сравнение существующих решений

Тут табличка с ранее описанными критериями сравнения.

Глава 2. Формулировка требований к решению

2.1 Функциональные требования

Разрабатываемый инструмент должен удовлетворять следующим функциональным требованиям:

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

2. Инструмент должен содержать возможность аутентификации пользователей, как по паре логин/пароль, так и через протокол LTI.

3. Должна существовать возможность разделения прав пользователей на администраторов и учащихся.

4. Администраторы должны иметь возможность создавать, редактировать и удалять задачи.

5. Каждая задача должна иметь: название, текст условия, связанный с ней чекер и его параметры.

6. Учащиеся должны уметь получать задания по конкретным задачам через системы управления обучением. Для этого администратор такой системы должен настроить LTI интеграцию, в частности, указать идентификатор нужной задачи.

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

8. Вместе с редактором кода должна быть доступна функциональность отладки: добавление и удаление точек останова, запуск, остановка, пошаговое исполнение программы. Если программа находится в приостановленном состоянии, должен быть доступен просмотр и редактирование регистров процессора и содержимого памяти процесса.

9. 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.

2.2 Нефункциональные требования

1. Инструмент, в первую очередь, предназначается для задач, направленных непосредственно на изучение языка ассемблера. Таким образом,

задачи должны быть составлены так, чтобы их решения могли исполняться в контексте непривилегированных процессов пользовательского пространства, без доступа к конкретным системным вызовам и периферии.

2. Инструмент должен минимизировать количество элементарных шагов, требуемых для запуска программ.

3. Инструмент должен быть эффективен по использованию процессорного времени и оперативной памяти.

4. Инструмент должен обладать достаточным быстродействием и отзывчивостью.

Глава 3. Описание метода решения

3.1 Компиляция ассемблерных программ

Используем GCC. Поддерживаются директивы `#line`, таким образом можно чинить номера строк с отладочной информации и сообщениях об ошибках.

3.2 Запуск ассемблерных программ

3.2.1 Изоляция с помощью seccomp

Seccomp — механизм ядра Linux, позволяющий процессу перейти в «безопасный режим», в котором запрещены все системные вызовы, кроме `exit`, `sigreturn`, `read` и `write`. Запретить работать со стандартными потоками ввода/вывода можно, предварительно вызвав `close` на них.

Здесь можно привести листинг кода, который переводит программу в безопасный режим. Код этот живёт в файле `environment/x86_64/entry.S`.

3.2.2 Ограничение потребляемых ресурсов с помощью Docker

Docker (через механизм контрольных групп) позволяет ограничивать использование процессорного времени, используемую память в контейнерах.

Существует системный вызов `setrlimit`. Ограничиваем процессорное время в секундах через `RLIM_CPU`. Docker умеет это делать через параметр `ulimit`.

3.3 Отладка ассемблерных программ

3.3.1 Использование GDB

GDB — консольный инструмент отладки программ. Позволяет отлаживать программы на самых разных языках программирования на разных платформах. Поддерживает отладочную информацию в формате DWARF.

3.3.2 GDB/MI

GDB/MI (GDB Machine Interface, машинный интерфейс GDB) позволяет взаимодействовать с процессом отладки в машиночитаемом виде. Это нам пригодится.

Тут можно вкратце описать формат взаимодействия, сослаться на мануал GDB/MI.

3.3.3 GDB server

GDB server — программа, с которой GDB может взаимодействовать, чтобы организовать отладку кода на удалённой машине. Пригодится для разделения полномочий и ограничения ресурсов.

3.4 Взаимодействие с пользователем через веб-браузер

Для интерактивной отладки 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, который позволяет общаться клиенту и серверу полностью асинхронно, а не по модели запрос-ответ.

3.5 Взаимодействие с системами управления обучением

Существует такой протокол: LTI, Learning Tools Interoperability. В частности, позволяет по протоколу OAuth авторизовывать пользователей конкретной LMS, получать информацию о таске, а также отправлять результаты проверки как score от 0.0 до 1.0.

3.6 Сбор метрик

Поднимаем Prometheus, пишем туда нужные метрики. Идеи для релевантных метрик: задержка исполнения команд gdb, время реакции на команды пользователя, общее потребление памяти на процесс отладки (gdb + gdbserver + программа), потребление cpu на процесс отладки (интересует idle cpu usage).

Глава 4. Архитектура программной организации

4.1 Требования к архитектуре

Хотим поддержать несколько вещей: интерактивную отладку, неинтерактивную проверку решений, интеграцию с LTI, админский интерфейс и прочие неинтересности. Поэтому имеет смысл выделить сервис, который будет заниматься именно взаимодействием с GDB.

4.2 Использованные технологии

4.2.1 Flask

Используем flask, потому что много плагинов для самых разных задач, необходимых в проекте: аутентификация, LTI, админка.

4.2.2 Asyncio + aiohttp

Используем asyncio и aiohttp в раннере, потому что хотим асинхронности и производительности.

4.2.3 Postgresql

Используем postgres как слой персистентности, потому что хотим надёжности и имеем нормализованные данные.

4.2.4 Redis

Используем redis для хранения nonceов, потому что хотим key-value хранилище с быстрым доступом и автоматической очисткой.

4.2.5 Nginx

Используем nginx, потому что хотим балансер, который позволяет направлять разные урлы на разные апстримы.

4.2.6 Docker

Потому что это всё надо изолировать же как-то. Docker compose для того чтобы не сойти с ума со всеми этими сервисами.

4.3 Схема архитектуры

Здесь нужна схема всего этого безобразия, она в целом такая: за балансером живёт статика, фласк и раннер. Фласк общается с БД и раннером, раннер сам по себе живёт, создаёт и убивает контейнеры с запущенными программами. Prometheus опрашивает стат-ручки в раннере и фласке.

4.4 Интерфейс пользователя

4.4.1 Аутентификация через логин и пароль

Есть у нас страница `/login`. Ожидается, что так входят только админы.

4.4.2 Аутентификация через LTI

При правильно настроенном Moodle, достаточно зайти на страницу задания и увидеть iframe с интерфейсом пользователя, вход происходит автоматически. Под капотом нам POST запрос к ручке `/lti` с нужными параметрами.

Скриншот мудла с iframeом.

4.4.3 Работа с интерактивным отладчиком через GUI

Можно ставить брейкпоинты, можно слать команды, можно смотреть на регистры. Обратите вниманеи на красивый редактор. Скриншот в paused состоянии.

4.4.4 Работа с админской панелью

Есть несколько сущностей: пользователи, задачи, задания и посылки, с ними работа ведётся одинаково.

4.5 Модель данных

4.5.1 Схема базы данных

Тут схема, которую как-то надо нарисовать.

4.5.2 Таблица users

Содержит в себе данные о пользователях, в том числе и метод аутентификации (password или LTI).

4.5.3 Таблица problems

Содержит информацию о задачах, в том числе и чекер.

4.5.4 Таблица assignments

Задания, выданные студентам. Содержит, помимо прочего, LTI callback и оценку.

4.5.5 Таблица submissions

Содержит посылки по задачам.

Глава 5. Исследование свойств решения

Выводы

Ну вот написали инструмент, все задачи поставленные во введении сделали, студент заслуживает оценки «отлично» и присвоения степени бакалавра.

Заключение

Чем это отличается от выводов? Не знаю, но шаблон СПбГУ требует.

Список литературы