Санкт-Петербургский государственный университет

АСЕЕВА Серафима Олеговна

Выпускная квалификационная работа

Разработка отладчика для IDE RuC

Уровень образования: бакалавриат

Направление *02.03.03 «Математическое обеспечение и администрирование информационных систем»*

Основная образовательная программа *СВ.5006.2018 «Математическое обеспечение и администрирование информационных систем»*

Профиль *Системное программирование*

Научный руководитель:
проф. каф. СП, д.ф.-м.н., проф. А.Н. Терехов

Рецензент:
руководитель проектов ООО "Софтком" Д.В. Тимохин

Санкт-Петербург
2022

Saint Petersburg State University

Serafima Aseeva

Bachelor's Thesis

Debugger for RuC IDE

Education level: bachelor

Speciality *02.03.03 "Software and Administration of Information Systems"*

Programme *CB.5006.2017 "Software and Administration of Information Systems"*

Profile: *System Programming*

Scientific supervisor:
Sc.D, prof. A.N. Terekhov

Reviewer:
project manager at "Softcom" D.V. Timokhin

Saint Petersburg
2022

Оглавление

1. Введение

Язык C — один из самых широко распространенных языков программирования. Он позволяет писать высокопроизводительные программы, он универсален и применяется в совершенно разных областях. Синтаксис языка C и структура программ на нем достаточно просты для понимания, но, несмотря на это, язык не вполне подходит для начинающих программистов: вероятность ошибки высока, а обнаружить эту ошибку с помощью стандартных средств разработки бывает очень трудно. Выход за границы массива, обращение к недоступной памяти или разыменование нулевого указателя - одни из наиболее частых примеров таких ошибок.

РуСи — язык, берущий за основу язык C и решающий некоторые из вышеописанных проблем [10]. Первоначально язык создавался как инструмент для обучения программированию, но впоследствии им заинтересовались разработчики высоконадежных систем, например, в области военной промышленности. Помимо стандартных (английских) ключевых слов языка C, были добавлены аналогичные им ключевые слова на русском языке, а также возможность использования кириллицы в именах идентификаторов и русскоязычные сообщения компилятора об ошибках. Такие изменения помогут в обучении программированию учеников младшей и средней школы, которые еще не знают английский язык достаточно хорошо.

Помимо этого, были добавлены некоторые ограничения, значительно повышающие безопасность программ и защищающие от типичных ошибок. Так, в языке РуСи отсутствует арифметика указателей; добавлен новый тип указателя — Never Null Pointer, защищающий пользователя от ошибочного разыменования нулевого указателя; улучшены инструменты работы с массивами и добавлена проверка на выход за границы массивов. Все эти изменения не только делают язык более простым для изучения начинающими программистами, но и позволяют использовать его в системах, где высокая надежность и защищенность от ошибок является критически важной.

Отладчик — инструмент, который знаком и необходим каждому программисту. Отладка программы позволяет исполнять программу пошагово, останавливаясь после каждой инструкции, и таким образом локализовывать и исправлять ошибки в ней. Именно поэтому в рамках данной дипломной работы разрабатывается отладчик для языка РуСи.

2. Постановка задачи

Цель данной работы — разработка отладчика для программ, написанным на языке РуСи и добавление к IDE возможности отладки. Для достижения этой цели были поставлены следующие задачи:

1. Изучение архитектуры транслятора РуСи.
2. Доработка модулей генерации и интерпретации кода для работы с отладочной информацией.
3. Реализация основных функций отладчика.
4. Тестирование.

3. Обзор

3.1. Существующие отладчики

Поскольку язык РуСи берет за основу язык С, имеет смысл рассмотреть, какие функции предоставляют наиболее популярные отладчики для этого языка.

1. GDB — кросс-платформенный отладчик проекта GNU, позволяющий отлаживать программы на языках C, C++ и множестве других языков [3]. GDB предоставляет обширную функциональность, такую как:

   (a) Работа с точками останова. Отладчик позволяет установить точку на выбранной строке, чтобы программа прерывалась, доходя до оператора в данной строке.

   (b) Условные точки останова - прерывание работы программы при выполнении заданного условия.

   (c) Трассировка - пошаговое исполнение программы, с возможностью остановки на каждой инструкции.

   (d) Просмотр текущих значений переменных в области видимости в точке остановки программы, а также значений регистров.

   (e) Просмотр стека вызовов.

   (f) Работа с многопоточными программами, переключение между потоками, просмотр и отслеживание значений переменных в каждом из потоков.

   Полный туториал и список команд можно найти на официальном сайте GNU [7].

2. LLDB — более современный и высокопроизводительный отладчик, использующий библиотеки проекта LLVM, такие как, например, парсер языка C CLang. В отличие от GDB, имеющего свободную

структуру команд, команды в LLDB имеют строго заданный синтаксис:
`<noun> <verb> [-options [option-value]] [argument [argument...]]`
Многие команды GDB имеют свои аналоги в LLDB - на официальном сайте проекта опубликован туториал по использованию LLDB для пользователей, знакомых с GDB [4]. Одно из значительных нововведений - LLDB позволяет изменять значения переменных прямо во время отладки с помощью команды expression. Перекомпиляцию кода при этом выполнять не нужно.

3. Первый вариант отладчика для RuC IDE был представлен в дипломной работе студентки кафедры системного программирования Дмитриевой Д.А. [8] В работе была описана архитектура отладчика, разработаны функции для работы с точками останова и добавлена функция остановки по доступу к переменной. Эта функция, не представленная в других отладчиках для языка C, представляет собой прерывание программы в момент, как только она получает доступ к нужной переменной. Однако результаты этой работы не вполне устроили научного руководителя и создателя языка РуСи Терехова А.Н. Некоторые желаемые функции (например, просмотр значений переменных в области видимости) не были реализованы, а некоторые, по мнению руководителя, требуют доработки. Поэтому данная дипломная работа является продолжением упомянутой работы.

3.2. Хранение отладочной информации

За основу реализации взят формат хранения отладочной информации DWARF [1]. Это стандарт хранения отладочной информации для процедурных компилируемых языков, таких как C, C++, Java, Fortran. На этапе генерации исполняемого кода создается файл, в который записывается блочная структура программы и локальные переменные, а также соответствие между строками исходного кода и операторами исполняемого кода. Поскольку в данный момент ведется активная раз-

работка транслятора РуСи на LLVM, использование стандартизированного формата представления отладочной информации будет полезно в дальнейшем.

3.3. Debug information entity

Debug Information Entity (или DIE) в формате DWARF - основная структура для представления отладочной информации. DIE используются для представления как отдельных переменных, так и блоков кода. Каждый DIE имеет тег, описывающий его тип, а также различные атрибуты - его имя/идентификатор, тип данных, адрес в памяти. DIE, описывающие крупный блок кода, имеют в качестве своих потомков DIEs, описывающие вложенные в него блоки и локальные переменные.

```
fig7.c:
        1:  int a;
        2:  void foo()
        3:  {
        4:      register int b;
        5:      int c;
        6:  }

<1>:    DW_TAG_subprogram
          DW_AT_name = foo
<2>:    DW_TAG_variable
          DW_AT_name = b
          DW_AT_type = <4>
          DW_AT_location = (DW_OP_reg0)
<3>:    DW_TAG_variable
          DW_AT_name = c
          DW_AT_type = <4>
          DW_AT_location =
                (DW_OP_fbreg: -12)
<4>:    DW_TAG_base_type
          DW_AT_name = int
          DW_AT_byte_size = 4
          DW_AT_encoding = signed
<5>:    DW_TAG_variable
          DW_AT_name = a
          DW_AT_type = <4>
          DW_AT_external = 1
          DW_AT_location = (DW_OP_addr: 0)
```

Рис. 1: Пример описания простого кода - из туториала с официального сайта стандарта DWARF [2]

3.4. Транслятор РуСи

Транслятор языка РуСи [9] разделен на две части: компилятор и виртуальную машину.

Компилятор принимает на вход текст, написанный пользователем на языке РуСи, производит его синтаксический разбор (lexer.c) и строит по нему синтаксическое дерево (builder.c). Затем генератор кода (codegen.c) обходит синтаксическое дерево и генерирует по нему код, исполняемый на виртуальной машине РуСи. Этот код представляет из себя вектор `mem`, описывающий память виртуальной машины.

Виртуальная машина содержит интерпретатор кода (import.c), который пошагово выполняет набор машинных команд из памяти.

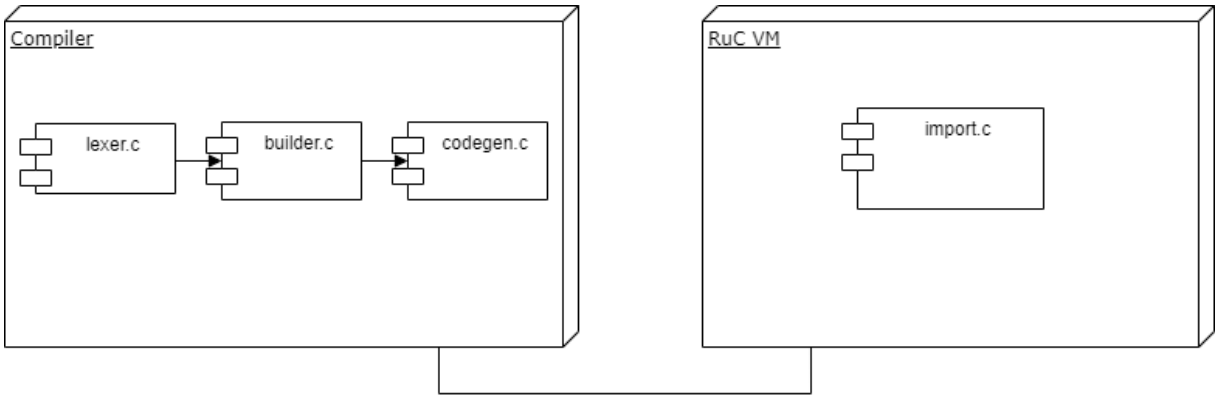


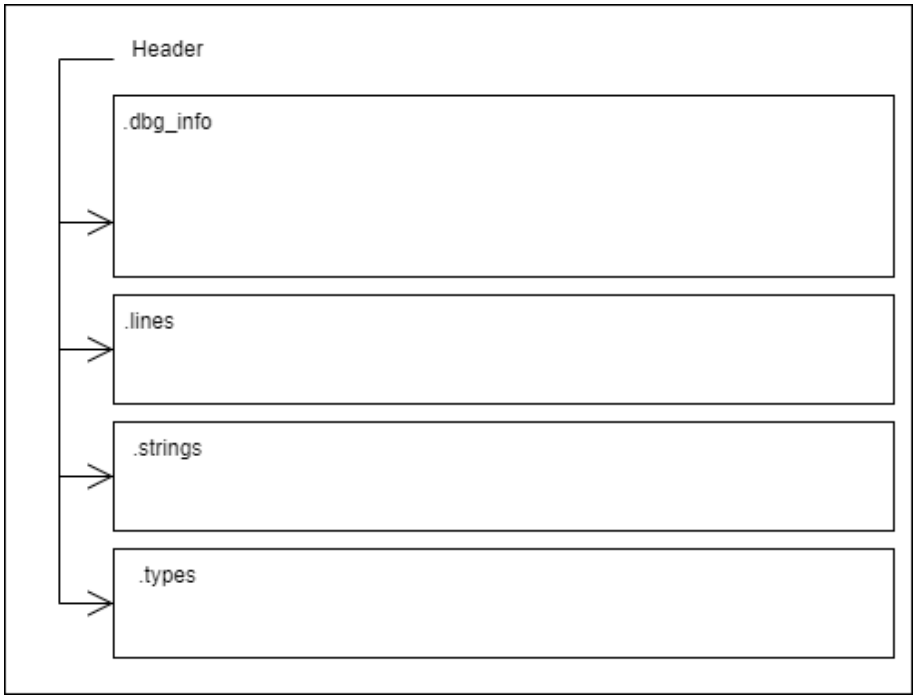
Рис. 2: Диаграмма компонентов транслятора РуСи.

4. Метод

4.1. Создание файла отладочной информации

Параллельно с генерацией кода для виртуальной машины генератор кода записывает отладочную информацию в файл dbg_info.bin. Данный файл состоит из:

1. Заголовка - содержит в себе указатели для доступа к секциям, на которые разделен данный файл.
2. .die_info - секция, которая хранит DIEs.
3. .line_table - секция, представляющая собой таблицу соответствия между строками пользовательского кода и адресами в виртуальной памяти, отсортированную по номерам строк.
4. .strings - секция, хранящая идентификаторы, использованные в программе.
5. .types - секция, описывающая использованные типы данных, их размер и представление.



Для генерации и чтения файлов были созданы модули debug_write.c и debug_read.c соответственно, а также внесены необходимые изменения в процедуры интерпретации и генерации кода.

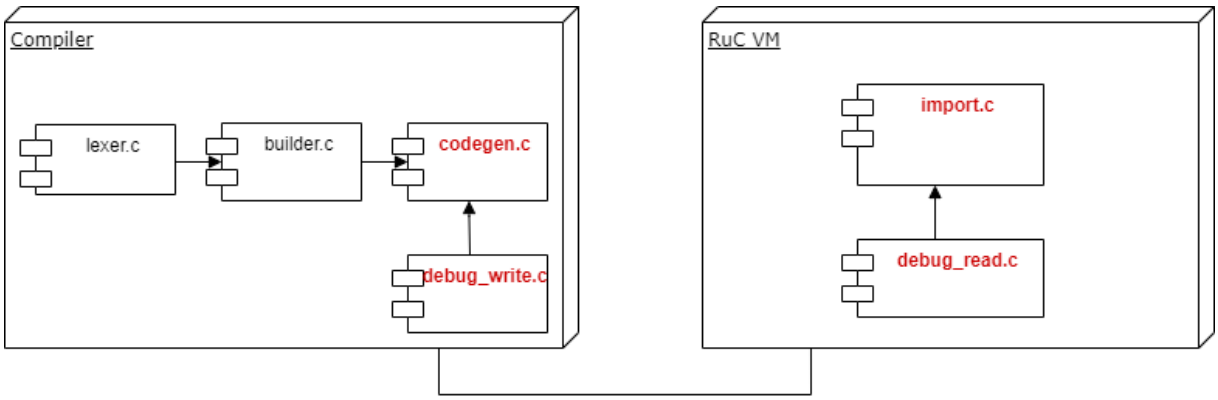


Рис. 3: Обновленная диаграмма компонентов.

4.2. Остановка по точке останова и трассировка

Для возможности отладки кода необходимо включить флаг отладки в вируальной машине. Если значение данного флага равно TRUE, интерпретатор находит в таблице строки, на которых были проставлены точки останова, и определяет адреса операторов, на которых нужно остановиться. При выполнении каждого оператора проверяется его адрес и, если на этом месте была поставлена точка останова, исполнение программы приостанавливается.

Трассировка - пошаговое (построчное) выполнение программы. Для реализации этого режима используется такой же алгоритм, где мы полагаем, что точка останова поставлена в каждой строке программы.

4.3. Поиск переменных и функций

Информация, записанная в секции .die_info представляет собой дерево, отображающее структуру отлаживаемой программы. Узлы этого дерева могут представлять собой переменные, типы переменных, выделенные блоки кода, функции или подпрограммы.

Для узла, описывающего подпрограмму, локальные переменные и вложенные подпрограммы являются его потомками.

Во время исполнения программы отладчик отслеживает текущий исполняемый блок кода и с помощью дерева составляет таблицу идентификаторов, видимых в данном блоке. Таким образом, пользователь может получить значение необходимой ему переменной по ее идентификатору.

5. Тестирование и апробация

Первый этап тестирования - модульное тестирование компонентов отладчика. Была протестирована корректность работы методов записи и чтения из файла, методов поиска строк и переменных.

Второй этап - апробация функций отладчика (остановки по номеру строки и просмотра значений переменных) на основе набора тестов, разработанных специально для языка RuC [11]. Апробация проведена для программ, содержащих различные языковые конструкции (вычисление простых арифметических выражений, объявление переменных, условные операторы, функции). Отдельно протестирована работа программ на латинице и кириллице.

6. Заключение

В ходе работы были выполнены следующие задачи:

1. Изучена архитектура транслятора и интерпретатора РуСи.
2. Внесены изменения в генератор кода и интерпретатор РуСи, необходимые для работы с отладочной информацией.
3. Реализована возможность остановки по точке останова и трассировка, а также просмотр значений переменных.
4. Выполнено тестирование отладчика.
5. Результаты работы загружены на GitHub [5] [6].

Список литературы

[1] Committee. DWARF Standards. The DWARF Debugging Standard. — URL: https://dwarfstd.org/ (online; accessed: 28.01.2022).

[2] Eager Michael J. Introduction to the DWARF Debugging Format. — URL:

[3] GNU. Project. GDB: The GNU Project Debugger. — URL: https://www.gnu.org/software/gdb/ (online; accessed: 28.01.2022).

[4] LLDB. Team The. LLDB Tutorial. — URL: https://lldb.llvm.org/use/tutorial.html (online; accessed: 28.01.2022).

[5] RuC. — URL: https://github.com/manelyset/RuC (online; accessed: 26.05.2022).

[6] RuC-VM. — URL: https://github.com/manelyset/RuC-VM (online; accessed: 26.05.2022).

[7] Shebs Richard Stallman Roland Pesch Stan. Debugging with GDB. — URL: https://ftp.gnu.org/old-gnu/Manuals/gdb/html_node/gdb_toc.html (online; accessed: 28.01.2022).

[8] Д.А. Дмитриева. Разработка отладчика для языка РуСи. // Кафедра системного программирования СПбГУ, 2017.

[9] Терехов А.Н. Терехов М.А. Инструментальное средство обучения программированию и технике трансляции. // Компьютерные инструменты в образовании, №1. — URL: http://cte.eltech.ru/ojs/index.php/kio/article/view/1388 (online; accessed: 26.05.2022).

[10] Терехов А.Н. Терехов М.А. Проект РуСи для обучения и создания высоконадежных программных систем. — Известия высших учебных заведений. Северо-Кавказский регион. Технические науки. // Cyberleninka. — URL: https://cyberleninka.ru/article/n/

proekt-rusi-dlya-obucheniya-i-sozdaniya-vysokonadezhnyh-programm
(online; accessed: 28.01.2022).

[11] Тесты для кодогенератора RuC. — URL: https://github.com/andrey-terekhov/RuC/tree/master/tests/codegen/executable (online; accessed: 26.05.2022).