



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«КАЗАНСКИЙ ГОСУДАРСТВЕННЫЙ ЭНЕРГЕТИЧЕСКИЙ УНИВЕРСИТЕТ»

Институт ЦТЭ
Кафедра ИТИС

О Т Ч Е Т
по производственной практике
(проектной)

Бобровского Сергея Денисовича,

обучающийся в группе ТРП-1-21 по образовательной программе

Технологии разработки программного обеспечения
направления подготовки
09.03.01 Информатика и вычислительная техника
указывается код и наименование направления подготовки

ОТЧЕТ ПРОВЕРИЛ

Руководитель практики

доц. каф. ИТИС Хуснутдинов Р.М.

«__» _____ 2024г.

Представитель профильной организации

Исмагилов И.Р.

ОЦЕНКА при защите отчета:

«__» _____ 2024 г.

Казань, 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
1 ИЗУЧЕНИЕ ПОДХОДОВ К ПРОЕКТНОЙ РАБОТЕ В IT-ИНДУСТРИИ	6
2 ИЗУЧЕНИЕ ЛИТЕРАТУРЫ ПО ИНТЕГРАЦИИ ЯП C++ И LUA В ОС LINUX	11
3 ИНТЕГРАЦИЯ LUA В C++	12
3.1 Разработка и создание вспомогательного инструмента, осу- ществляющего конвертацию скрипта Lua в файл исходного кода C++	12
3.1.1 Обоснование необходимости	12
3.1.2 Разработка и создание вспомогательного инструмента .	12
3.2 Проектирование приложения	14
3.2.1 Средства разработки	14
3.2.2 Дерево файлов приложения	15
3.2.3 Механизм сборки приложения	15
3.3 Разработка функционала приложения	17
3.4 Тестирование работы приложения	19
3.4.1 Сборка приложения	19
3.4.2 Запуск приложения по сценарию 1	21
3.4.3 Запуск приложения по сценарию 2	25
4 ИНТЕГРАЦИЯ C++ В LUA	27
4.1 Сборка	27
4.2 Тестирование	30
ВЫВОД	31
БИБЛИОГРАФИЧЕСКИЙ СПИСОК	32
ПРИЛОЖЕНИЯ	33

Файл проекта NodeJS – package.json	33
Файл ключа – key.js	34
Основной исполняемый файл – compile.js	35
Файл проекта – CMakeLists.txt	37
Основной файл проекта C++ – main.cpp	39
Заголовочный файл экспортируемых в Lua функций – LuaFunctions.h	42
Файл исходного кода экспортируемых в Lua функций – LuaFunctions.cpp	43
Скрипт Lua – Startup.lua	46
Файл проекта обратной интеграции – CMakeLists.txt	47
Основной файл проекта обратной интеграции C++ – main.cpp	48

ВВЕДЕНИЕ

Производственная практика (проектная) направлена на приобретение навыков и опыта самостоятельного решения практических задач, закрепление и углубление теоретической подготовки обучающегося, а также развитие практических навыков и компетенций, необходимых для работы с современными языками программирования и инструментами разработки.

Актуальность данной работы обусловлена растущей потребностью в эффективной интеграции различных языков программирования в условиях стремительного развития технологий. Одной из таких актуальных задач является интеграция языков C++ и Lua, что позволяет создавать мощные и гибкие решения для различных областей применения. Возможность взаимодействия C++ и Lua предоставляет разработчикам инструменты для расширения функциональности приложений, их оптимизации и упрощения разработки за счет использования скриптовых возможностей Lua.

Целью данной практики является изучение механизмов интеграции языков программирования C++ и Lua в среде операционной системы Linux, а также разработка инструмента, который будет осуществлять конвертацию Lua-скриптов в файлы исходного кода C++.

Задачами производственной (проектной) практики являются:

- изучение литературы по интеграции C++ и Lua, а также технологий, используемых для этой цели;
- разработка вспомогательного инструмента для конвертации Lua-скриптов в C++-код;
- выполнение двухсторонней интеграции Lua и C++ с использованием IDE VSCodium и системы сборки CMake;
- разработка программного кода, демонстрирующего успешную интеграцию и взаимодействие двух языков.

Данная работа способствует закреплению теоретических знаний и навыков проектирования информационных процессов, а также приобретению

практического опыта в области программирования и интеграции современных технологий.

1 ИЗУЧЕНИЕ ПОДХОДОВ К ПРОЕКТНОЙ РАБОТЕ В IT-ИНДУСТРИИ

Методологии разработки ПО представляют собой совокупность определенных подходов, принципов и инструментов, которые позволяют управлять проектами разработки для достижения поставленных целей [1]. Основные цели внедрения методологий разработки ПО:

- увеличение эффективности ПО;
- улучшение качества ПО;
- достижение высокого уровня прозрачности и управляемости проекта;
- улучшение коммуникации и сотрудничества;
- достижение гибкости и адаптивности.

Ниже перечислены основные методологии разработки ПО:

1. **Waterfall Model** (каскадная модель или «водопад»). Каскадная модель разработки программного продукта — это классический подход к разработке ПО, который предполагает линейное выполнение последовательных этапов разработки: сбор требований, проектирование, реализацию, тестирование и сопровождение. Каждый этап начинается только после успешного завершения предыдущего, то есть в этой модели нет возможности вернуться на предыдущие этапы. В каскадной модели каждый этап разработки имеет свою документацию и четкие требования, которые должны быть выполнены, прежде чем можно перейти к следующему этапу. Это делает каскадную модель более предсказуемой и позволяет лучше контролировать процесс разработки, особенно в случаях, когда требования к продукту уже четко определены. Однако, такой линейный подход также имеет свои недостатки, например, сложность изменения требований на поздних стадиях проекта или необходимость повторения предыдущих этапов при обнаружении ошибок. Кроме того, каскадная модель не учитывает возможности быстро реагировать на изменения, которые могут произойти в процессе разработки.

2. **V-модель** разработки программного продукта — это модель, которая является улучшенной версией каскадной модели. В V-модели процесс разработки представлен в виде буквы «V», где каждый этап проекта имеет соответствующий этап тестирования. Основная идея V-модели заключается в том, что качество продукта зависит от качества его тестирования. Каждый этап разработки имеет соответствующий этап тестирования, начиная с тестирования требований и заканчивая тестированием внедрения. Преимуществом V-модели является то, что она помогает обнаружить ошибки и дефекты на ранних этапах разработки, что может существенно сократить время и затраты на доработку продукта. Это также делает процесс разработки более предсказуемым и улучшает контроль качества. Однако, также как и каскадная модель, V-модель не предусматривает возможности быстро адаптироваться к изменениям в требованиях к продукту, что может ограничивать гибкость разработки в динамичных средах. В таких случаях лучше использовать более гибкие методологии, такие как Agile или Scrum.

3. **Agile** — это философия разработки программного обеспечения, которая призывает к гибкости и адаптивности в процессе разработки продукта. Она акцентирует внимание на быстрой итеративной разработке продукта с частыми поставками рабочего программного кода, которые постоянно улучшаются на основе обратной связи от пользователей и заказчиков. Методологии Agile, такие как Scrum, Kanban, Extreme Programming (XP), Feature Driven Development (FDD) и другие, позволяют командам быстро реагировать на изменения требований и перестраивать свой подход к разработке, чтобы достичь лучших результатов. В центре философии Agile лежит идея о том, что процесс разработки должен быть гибким и адаптивным, что позволяет командам быстро реагировать на изменения и принимать важные решения на основе обратной связи от пользователей и заказчиков. В Agile-разработке большое внимание уделяется коммуникации внутри команды и с заказчиками, автоматизации тестирования, постоянной интеграции и обновлению кода, улучшению процессов и инструментов разработки, а также нахождению наилучших практик и оптимизации всего процесса разработки. Благодаря этому подходу Agile-разработка

позволяет снизить время и стоимость разработки, увеличить качество и улучшить удовлетворенность пользователей.

4. **RAD** (Rapid Application Development) — разновидность инкрементной модели, которая акцентирует внимание на быстрой разработке продукта в условиях сильных ограничений по срокам и бюджету и нечётко определённых требований к продукту. Эффект ускорения разработки достигается путём непрерывного, параллельного с ходом разработки, уточнения требований и оценки текущих результатов с привлечением заказчика.

5. **Модель Spiral** (спиральная модель) — это гибкая методология разработки программного обеспечения, которая сочетает в себе итеративный подход с последовательностью шагов, основанных на рисках. Основная идея спиральной модели заключается в том, чтобы разбить проект на более мелкие итерации, каждая из которых содержит этапы планирования, анализа рисков, проектирования, реализации, тестирования и оценки. Каждая итерация зависит от предыдущей и планируется на основе ее результатов. Спиральная модель включает в себя оценку рисков на каждой стадии проекта, а также их управление. Это помогает снизить риски и улучшить качество продукта, который создается в процессе разработки. В данном проекте используется методология разработки Scrum (Рисунок 1.1), основанная на принципах Agile [2]. Проект разделяется на небольшие задачи, которые необходимо сделать за определенное количество времени (эти временные интервалы, как и в Agile, называются спринтами). Модель разработки ПО Scrum построена таким образом, чтобы помочь командам естественным образом адаптироваться к меняющимся условиям рынка и потребностям пользователей. В то же время короткие циклы позволяют разработчикам быть эффективнее. Scrum обеспечивает структуру, оптимизирует разработку и при этом остается гибким и учитывает желания владельца продукта.



Рисунок 1.1 – Схема методологии Scrum

Разработка программного обеспечения на C++ в операционной системе Linux с использованием CMake — это современный подход к созданию кроссплатформенных приложений, который позволяет упрощать управление проектом и процесс сборки. CMake является мощным инструментом для автоматизации сборки, настройки параметров компиляции, интеграции библиотек и организации структуры кода [3][4].

CMake работает как генератор файлов сборки, создавая конфигурации для таких инструментов, как Make, Ninja или Visual Studio. Это делает его идеальным выбором для проектов на C++, которые требуют поддержки разных платформ и конфигураций [4]. Основные этапы разработки на C++ с использованием CMake:

1. **Инициализация проекта.** Проект начинается с создания файла CMakeLists.txt, который описывает настройки сборки, исходные файлы и зависимости. Этот файл указывает минимальную версию CMake, имя проекта и исполняемый файл, который будет создан [4].

2. **Создание сборочной директории.** Принято разделять исходный код и файлы сборки, используя отдельную папку. Этот подход упрощает управление проектом, так как все временные файлы находятся в одном месте [3].

3. **Добавление библиотек.** Одним из ключевых преимуществ CMake является простое подключение внешних библиотек. Это делает проект гибким и позволяет легко интегрировать дополнительные модули [4].

4. **Тестирование и настройка.** CMake поддерживает автоматизированное тестирование, что особенно важно для больших проектов. Инструменты вроде CTest упрощают создание тестов и интеграцию их в процесс сборки [4].

5. **Оптимизация.** Для улучшения производительности на этапе компиляции можно включить оптимизационные флаги.

Преимущества использования CMake:

- Кроссплатформенность: CMake поддерживает Windows, Linux и macOS, что позволяет разрабатывать универсальные приложения [3].
- Модульность: Удобная интеграция внешних библиотек и модулей.
- Автоматизация: Упрощение процессов сборки и тестирования [4].

2 ИЗУЧЕНИЕ ЛИТЕРАТУРЫ ПО ИНТЕГРАЦИИ ЯП C++ И LUA В ОС LINUX

Интеграция языков программирования C++ и Lua является важным вопросом, потому что: C++ является одним из самых популярных языков для создания высокопроизводительных приложений, в то время как Lua представляет собой легковесный, высокоэффективный скриптовый язык, который активно используется для расширения функциональности и настройки программ.

В рамках интеграции Lua в C++ приложения особое внимание уделяется таким библиотекам, как LuaBridge, которая упрощает взаимодействие между этими языками. LuaBridge 3.0 предоставляет обширную документацию и примеры использования, что делает процесс интеграции удобным и доступным для разработчиков [6]. Данная библиотека не является единственной в своем роде, и имеет аналоги [7].

В статье «Что такое скрипты и с чем их едят» на платформе Habr, объясняется принцип работы с Lua в контексте разработки скриптов, а также описываются основные возможности и области применения скриптовых языков в C++ проектах, в том числе в операционных системах Linux. Lua предлагает простоту в использовании и гибкость, что делает его популярным выбором для внедрения в C++ приложения для динамической настройки и расширения функционала [5].

Причем в приведенной статье рассматривается именно работа с «голым» Lua, что предпочтительно.

3 ИНТЕГРАЦИЯ LUA В C++

3.1 Разработка и создание вспомогательного инструмента, осуществляющего конвертацию скрипта Lua в файл исходного кода C++

3.1.1 Обоснование необходимости

Данный шаг, строго говоря, не является обязательным. Однако дает ряд неоспоримых преимуществ:

- Компактность. Программу можно будет поставлять в виде одного файла (особенно при условии статической компоновки).
- Гибкость и удобство. К работе будет предоставлен не файл, который надо читать с диска, проверять наличие, ошибки и т.п, а строка в готовом и родном для C виде (массив символов). В таком случае, ошибки будут отсеиваться на этапе компиляции.
- Безопасность. Файл скрипта можно будет зашифровать, сделав код всей программы закрытым. За счет этого, он будет так же труден для изменения и чтения, как и обычная скомпилированная программа на C.

3.1.2 Разработка и создание вспомогательного инструмента

В качестве средств разработки были выбраны:

- JS – скриптовый язык программирования.
- VSCodium – IDE.
- NodeJS – интерпретатор языка JavaScript.
- js-crypto-aes – библиотека для работы с криптографией.

Сам проект представляет собой проект на NodeJS (Листинг 4.2). Состоит из двух исполняемых файлов:

- key.js (Листинг 4.2) – файл с ключом AES.

- `compile.js` (Листинг 4.2) – основной файл, осуществляющий преобразование скриптов Lua в удобный для работы в C++ формат; шифрование исходного Lua-кода, для его последующей расшифровки уже во время исполнения приложения на C++.

Помимо преобразования Lua файлов, проект также осуществляет конвертацию ключа в формат, удобный для работы в C++.

Алгоритм работы программы:

1. Чтение файлов Lua: Она ищет все файлы с расширением `.lua` в текущей директории.
2. Обработка содержимого Lua файлов:
 - Читает каждый Lua файл и преобразует его содержимое в строку C++ с добавлением экранированных символов для новой строки (`\n`).
 - Удаляет пустые строки и строки, начинающиеся с комментариев.
 - Преобразует фигурные скобки и запятые в формат, совместимый с C++.
3. Шифрование содержимого:
 - Преобразует строку в байты и шифрует с использованием AES (режим CBC) с заранее сгенерированным ключом и вектором инициализации (IV).
4. Запись зашифрованного скрипта в файл C++:
 - Записывает зашифрованный скрипт в C++ заголовочный файл в виде строки (с кодировкой байтов как).
 - Добавляет информацию о длине зашифрованного содержимого.
5. Запись AES ключа в файл C++:
 - Записывает сгенерированные значения для `salt`, `key`, и `iv` в отдельный C++ заголовочный файл.

3.2 Проектирование приложения

3.2.1 Средства разработки

Для разработки данного проекта были выбраны следующие средства:

1. VSCodium – IDE. Является OpenSource версией проприетарного редактора кода VSCode, позиционирующего себя, как легковесную среду разработки. Поддерживает множество языков и инструментов программирования, в том числе C++ и CMake, с выбором компилятора (CLang).
2. Соответствующие расширения для VSCodium: для C++, CMake, Lua и JavaScript.
3. CMake – кроссплатформенный инструмент для создания приложений, поддерживающий различные компиляторы. Связующее звено проекта.
4. CLang – компилятор языка C++. На данный момент более перспективен, чем GCC.
5. lua 5.4.4 – библиотека Lua для C++ [9].
6. plusaes – библиотека для работы с AES в C++.
7. foresteamnd – собственная библиотека, предоставляющая ряд утилитарных функций и упрощенных реализаций некоторых механизмов [8].

Стоит отдельно упомянуть, что оригинальный Lua написан на C, собирается через обычный makefile, да еще и в библиотеку динамической компоновки (.so). Это не только затруднило бы работу с ним, но и перечеркнуло портативность: пришлось бы поставлять библиотеку Lua вместе с приложением, либо рассчитывать, что у пользователя она уже установлена. Поэтому вместо оригинального Lua была взята библиотека, уже переписанная под сборку через CMake [9], которая, к тому же, умеет собираться статически.

3.2.2 Дерево файлов приложения

Директория проекта приложения имеет следующую структуру:

- include – Подключаемые файлы библиотек
- lib – Собранные файлы библиотек
- lua – Проект Lua 5.4.4 на CMake
- plusaes – Репозиторий библиотеки plusaes
- CMakeLists.txt – Основной файл проекта CMake
- src – Исходный код проекта
- lua – Исходный код скриптов lua, конвертированные в .h файлы этих скриптов, ключ AES, вспомогательный инструмент для конвертации скриптов в зашифрованные файлы C++.
- main.cpp – Основной файл приложения.
- LuaFunctions.h – Заголовочный файл экспортируемых в Lua функций.
- LuaFunctions.cpp – Файл, отвечающий за экспорт функций в Lua.
- build – Продукты сборки проекта.

3.2.3 Механизм сборки приложения

За сборку проекта отвечает файл CMakeLists.txt (Листинг 4.2). Данный файл представляет собой скрипт CMake. Этот CMake-скрипт организует сборку проекта, который использует C++ и Lua. Он включает компиляцию файлов Lua с помощью Node.js, создает исполняемый файл rut, который использует скомпилированные Lua-файлы, и настраивает различные платформы и зависимости для успешной сборки. Шаги сборки:

1. Название проекта: rut, v0.1.0.
2. Добавление подпроекта: lua
3. Добавление пользовательского продукта сборки – файлов C++, конвертированных из скриптов Lua, а также файла ключа.
4. Листинг файлов проекта, в порядке их компиляции и сборки.

5. Добавление продуктов сборки скриптов Lua как зависимостей проекта.
6. Добавление директорий включения (include).
7. Компоновка библиотек статических библиотек: `foresteamnd` и `lua_static`.

На выходе получается файл *rut*, и ряд побочных продуктов сборки, вроде `makefile` в директории `build` проекта.

3.3 Разработка функционала приложения

Функционал приложения включает:

1. расшифровку кода скриптов (скрипта) Lua;
2. инициализацию Lua;
3. регистрацию собственных функций, в том числе в пространствах имен;
4. запуск строки скрипта на исполнение.

Алгоритм работы в подробностях:

В начале программы происходит расшифровка скрипта, который был зашифрован для защиты или уменьшения размера. Для этого используется функция `Decrypt`, которая принимает зашифрованные данные (в данном случае скрипт Lua) и расшифровывает их с помощью алгоритма AES в режиме CBC.

Следующий этап – инициализация среды Lua с помощью `luaL_newstate`. Это создает новый объект `lua_State`, который представляет собой состояние Lua, в котором будут выполняться скрипты.

Далее регистрируются функции C++ в Lua, что позволяет Lua-скрипту вызывать их. В частности, функции, которые были объявлены в пространстве `LuaFunctions`, такие как функция таймера, выполнение команд в оболочке (`bash`) и другие полезные функции. Функции добавляются в Lua через `LuaFunctions::Register`.

После расшифровки и инициализации среды Lua, скрипт выполняется с помощью `luaL_dostring`. Эта функция интерпретирует и выполняет Lua-код, переданный в виде строки.

После того как скрипт выполнен, можно вызывать Lua-функции из C++. Для этого используется несколько шагов:

- Получение глобальной функции с помощью `lua_getglobal(L, "FunctionName")`. Это помещает функцию в стек Lua.
- Пуш аргументов (если необходимо) в стек Lua, используя, например, `lua_pushnumber` для числовых значений.

– Вызов функции с помощью `lua_pcall(L, numArgs, numResults, 0)`, где `numArgs` – это количество аргументов, передаваемых функции, а `numResults` – количество ожидаемых результатов.

– Чтение результата из стека Lua с помощью функций, таких как `lua_tonumber`, `lua_toboolean` и так далее.

Пример вызова Lua-функции `IsWindows` из C++:

```
// достаём Lua-функцию в C++
lua_getglobal(L, "IsWindows");
// вызываем функцию
lua_pcall(L, 0, 1, 0);
cout << "Am i on windows? " << (lua_toboolean(L, -1) ? "true" : "false") << endl;
lua_pop(L, 1);
```

Рисунок 3.1 – Вызов Lua-функции `IsWindows` из C++

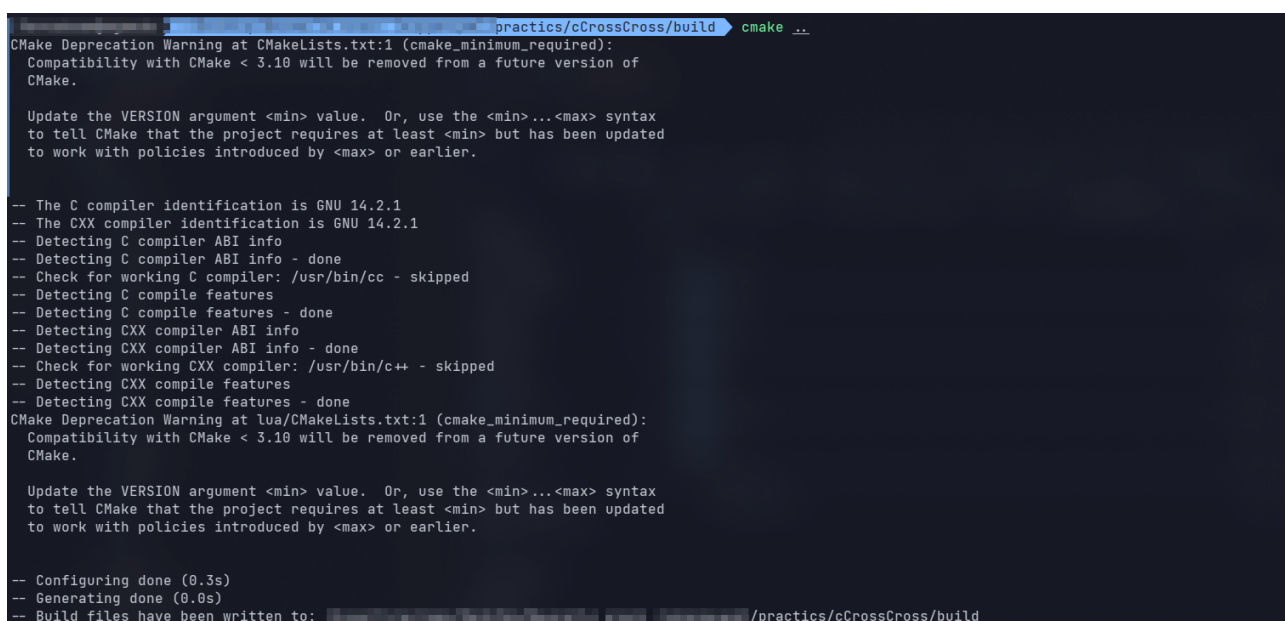
Завершающий этап – очистка. После завершения работы с Lua нужно закрыть состояние Lua, используя `lua_close(L)`, чтобы освободить все ресурсы, связанные с этим состоянием. Также удалить массив с расшифрованным скриптом.

3.4 Тестирование работы приложения

3.4.1 Сборка приложения

Сборка приложения выполняется в 2 этапа:

1. Переход в директорию сборки и создание файлов сборки (Рисунок 3.2).
2. Сборка приложения (Рисунок 3.3).



```
practices/cCrossCross/build cmake ..
CMake Deprecation Warning at CMakelists.txt:1 (cmake_minimum_required):
Compatibility with CMake < 3.10 will be removed from a future version of
CMake.

Update the VERSION argument <min> value. Or, use the <min>...<max> syntax
to tell CMake that the project requires at least <min> but has been updated
to work with policies introduced by <max> or earlier.

-- The C compiler identification is GNU 14.2.1
-- The CXX compiler identification is GNU 14.2.1
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
CMake Deprecation Warning at lua/CMakelists.txt:1 (cmake_minimum_required):
Compatibility with CMake < 3.10 will be removed from a future version of
CMake.

Update the VERSION argument <min> value. Or, use the <min>...<max> syntax
to tell CMake that the project requires at least <min> but has been updated
to work with policies introduced by <max> or earlier.

-- Configuring done (0.3s)
-- Generating done (0.0s)
-- Build files have been written to: /practices/cCrossCross/build
```

Рисунок 3.2 – Подготовка файлов сборки приложения

```

[ 0%] Built target luacompile
[ 2%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/lapi.c.o
[ 5%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/lcode.c.o
[ 7%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/lctype.c.o
[ 10%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/ldebug.c.o
[ 13%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/ldo.c.o
[ 15%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/ldump.c.o
[ 18%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/lfunc.c.o
[ 21%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/lgc.c.o
[ 23%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/llex.c.o
[ 26%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/lmem.c.o
[ 28%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/lobject.c.o
[ 31%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/lopcodes.c.o
[ 34%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/lparser.c.o
[ 36%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/lstate.c.o
[ 39%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/lstring.c.o
[ 42%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/ltable.c.o
[ 44%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/ltm.c.o
[ 47%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/lundump.c.o
[ 50%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/lvm.c.o
[ 52%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/lzio.c.o
[ 55%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/lauxlib.c.o
[ 57%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/lbaselib.c.o
[ 60%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/lcorolib.c.o
[ 63%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/ldblib.c.o
[ 65%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/liblib.c.o
[ 68%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/lmathlib.c.o
[ 71%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/loadlib.c.o
[ 73%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/loslib.c.o
[ 76%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/lsrplib.c.o
[ 78%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/lstolib.c.o
[ 81%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/lut8lib.c.o
[ 84%] Building C object lua/lua-5.4.4/CMakeFiles/lua_static.dir/src/luinit.c.o
[ 86%] Linking C static library liblua.a
[ 89%] Building CXX object CMakeFiles/rut.dir/src/LuaFunctions.cpp.o
[ 92%] Building CXX object CMakeFiles/rut.dir/src/main.cpp.o
[ 94%] Linking CXX executable rut
[ 94%] Built target rut
[ 97%] Building C object lua/lua-5.4.4/CMakeFiles/luac.dir/src/luac.c.o
[100%] Linking C executable ../luac
[100%] Built target luac

```

Рисунок 3.3 – Сборка приложения

3.4.2 Запуск приложения по сценарию 1

Для подготовки потребовалось:

1. Сделать идентичные функции поиска простых чисел в C++ и Lua.

```
bool IsPrime(int64_t n) {
    if (n < 2)
        return false;
    for (int i = 2; i <= sqrt(n); i++)
        if (n % i == 0)
            return false;
    return true;
}

int64_t lCPrimes(int64_t n) {
    int64_t primes = 0;
    for (int i = 2; i <= n; i++)
        if (IsPrime(i))
            primes = primes + 1;
    return primes;
}
```

Рисунок 3.4 – Функция поиска простых чисел в C++

```
function IsPrime(n)
    if n < 2 then
        return false
    end
    for i = 2, math.sqrt(n) do
        if n % i == 0 then
            return false
        end
    end
    return true
end

function LuaPrimes(n)
    local primes = 0
    for i = 2, n do
        if IsPrime(i) then
            primes = primes + 1
        end
    end
    return primes
end
```

Рисунок 3.5 – Функция поиска простых чисел в Lua

```

int LuaPrimes(lua_State* L, int number) {
    // получаем функцию
    lua_getglobal(L, "LuaPrimes");
    // помещаем аргумент в стек
    lua_pushnumber(L, number);
    // вызываем функцию
    lua_pcall(L, 1, 1, 0);
    double result = lua_tonumber(L, -1);
    // очищаем стек
    lua_pop(L, 1);

    return result;
}

```

Рисунок 3.6 – Функция запуска функции простых чисел из Lua в C++

2. Сделать в Lua функцию проверки операционной системы (Windows/Unix).

```

function IsWindows()
    return string.sub(os.getenv('HOME'), 1, 1) ~= '/'
end

```

Рисунок 3.7 – Функция проверки операционной системы на принадлежность к семейству Unix

3. Сделать в C++ команду для запуска программы из оболочки (bash), а также 2 команды для работы с таймером: его запуск с возвращением ключа для дальнейшего обращения, и его завершение, которое возвращает время, прошедшее с запуска. Добавить команду для показа системного окна уведомлений.

Далее, для тестирования работы, было произведено следующее:

1. Вызвана системная команда «neofetch», отображающая сведения о системе в вывод.

2. Показано системное окно с текстом (Рисунок 3.8).

3. Выведено, на каком семействе операционных систем запущена программа.
4. Вызвана C++ функция поиска простых чисел, выведен результат и затраченное время.
5. Далее, из C++ вызвана Lua-функция IsWindows, и так же выведен результат о семействе операционной системы.
6. Запущена из C++ Lua-функция поиска простых чисел, выведен ее результат и затраченное время.

Финальный результат работы: Рисунок 3.9.

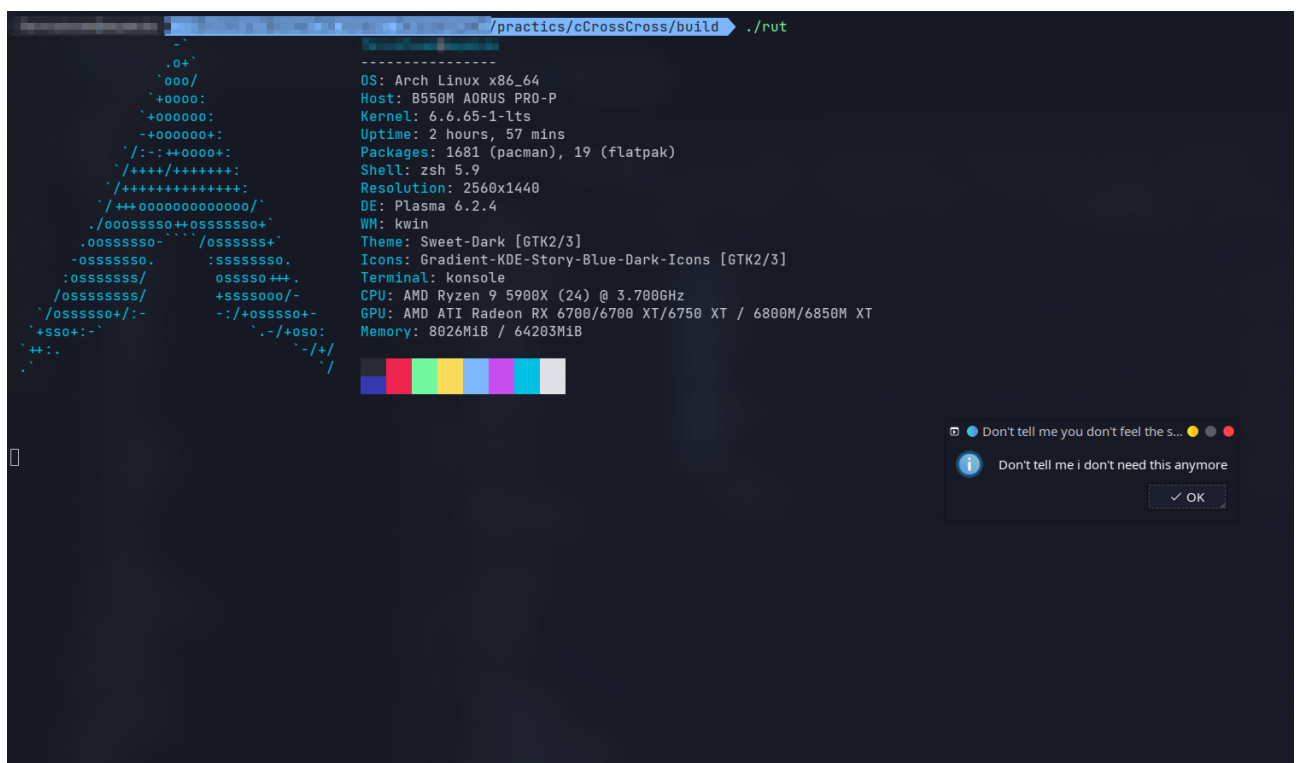


Рисунок 3.8 – Результат работы программы 1. Системное информационное
ОКНО

```

    .o+
    'ooo/
    '+oooo:
    '+oooooo:
    '+oooooo+:
    '/:-'+oooo+:
    '/+++/+++++:
    '/+++++++:
    '/++000000000000/'
    ./00000000++0000000+
    .0000000-./0000000+
    -0000000-:0000000.
    :0000000/ 00000++
    /0000000/ +000000/-
    /000000+/- -:/+00000+
    '+00+:-' -:/+000:
    '+:.. -:/+
    .:/

-----
OS: Arch Linux x86_64
Host: B550M AORUS PRO-P
Kernel: 6.6.65-1-lts
Uptime: 2 hours, 57 mins
Packages: 1681 (pacman), 19 (flatpak)
Shell: zsh 5.9
Resolution: 2560x1440
DE: Plasma 6.2.4
WM: kwin
Theme: Sweet-Dark [GTK2/3]
Icons: Gradient-KDE-Story-Blue-Dark-Icons [GTK2/3]
Terminal: konsole
CPU: AMD Ryzen 9 5900X (24) @ 3.700GHz
GPU: AMD ATI Radeon RX 6700/6700 XT/6750 XT / 6800M/6850M XT
Memory: 8026MiB / 64203MiB

on unix
CPrimes from Lua:      78498
Elapsed 0.277796042    sec
Am i on windows? false
LuaPrimes from C++: 78498
Elapsed: 1.81183 sec

```

Рисунок 3.9 – Результат работы программы 1. Вывод в консоль

3.4.3 Запуск приложения по сценарию 2

Запуск скрипта из скрипта.

Мы можем также использовать среду Lua, чтобы динамически запускать из нее сценарии. Для этого, добавим следующие функции в C++ и в Lua:

```
string RemoteExecReadline(lua_State* L, string command) {  
    lua_getglobal(L, "RemoteExec");  
    lua_pushstring(L, command.c_str());  
    lua_pcall(L, 1, 1, 0);  
    string result = "empty string";  
    if (lua_isstring(L, -1))  
        result = lua_tostring(L, -1);  
    lua_pop(L, 1);  
  
    return result;  
}
```

Рисунок 3.10 – Функция выполнения скрипта в C++

```
function RemoteExec(code)  
    return load(code, 'rExec')()  
end
```

Рисунок 3.11 – Функция выполнения скрипта в Lua

Теперь, получим сценарий от пользователя из консоли:

```
cout << "Enter a script to execute: ";  
string command;  
getline(cin, command);  
string output = RemoteExecReadline(L, command);  
cout << "Output: " << output << endl;
```

Рисунок 3.12 – Выполнение пользовательского сценарий в окружении Lua

Запустим программу:



Рисунок 3.13 – Результат работы программы 2. Системное информационное
ОКНО

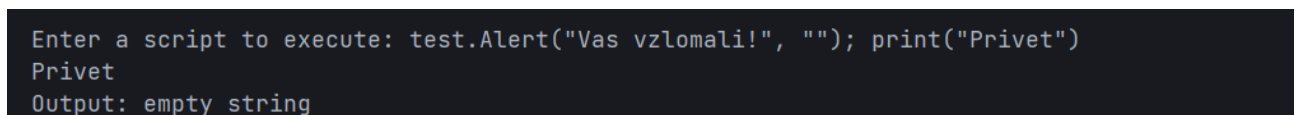


Рисунок 3.14 – Результат работы программы 2. Вывод в консоль

4 ИНТЕГРАЦИЯ C++ В LUA

4.1 Сборка

Помимо использования Lua из C++, что является более целесообразным, в виду скудности инструментария встроенных библиотек Lua, существует также возможность обратной интеграции. В этом случае, исходный код C++ компилируется в библиотеку (статической либо динамической компоновки), после чего подключается и используется из Lua.

Для осуществления этой обратной интеграции, структуру проекта нужно изменить следующим образом:

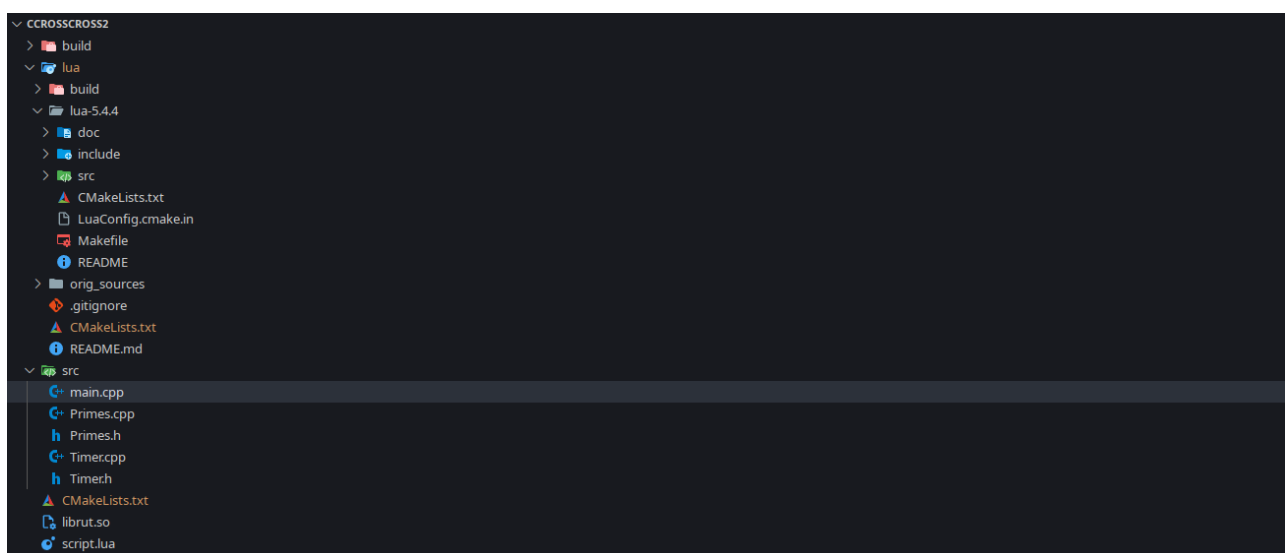


Рисунок 4.1 – Файловая структура проекта обратной интеграции

Основные изменения:

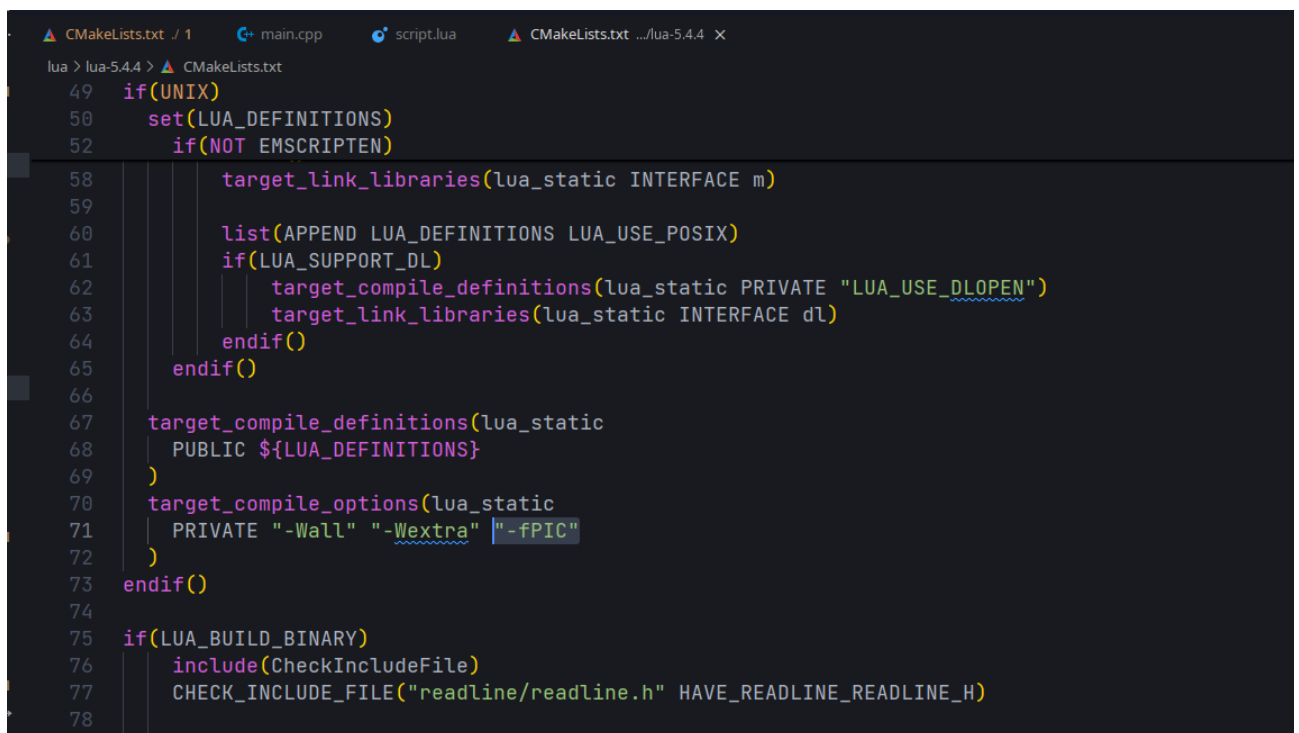
- Функции таймеров и поиска простых чисел вынесены в отдельные файлы.
- CMakeLists.txt был переписан для сборки проекта в динамическую библиотеку (.so).
- Проект будет собираться в библиотеку динамической компоновки, .so, а не в исполняемый файл.

– Вместо LuaFunctions теперь есть только main.cpp, который и управляет регистрацией функций библиотеки.

Основные изменения в Листинг 4.2:

– Строка с add_executable была изменена на add_library, был добавлен флаг SHARED, чтобы проект собирался в файл .so.

Также, чтобы проект собирался в .so, нужно было внести изменения в файл проекта библиотеки Lua, lua/lua-5.4.4/CMakeLists.txt, и добавить следующий флаг сборки:



```
lua > lua-5.4.4 > CMakeLists.txt
49  if(UNIX)
50      set(LUA_DEFINITIONS)
52      if(NOT EMSCRIPTEN)
58          target_link_libraries(lua_static INTERFACE m)
59
60          list(APPEND LUA_DEFINITIONS LUA_USE_POSIX)
61          if(LUA_SUPPORT_DL)
62              target_compile_definitions(lua_static PRIVATE "LUA_USE_DLOPEN")
63              target_link_libraries(lua_static INTERFACE dl)
64          endif()
65      endif()
66
67      target_compile_definitions(lua_static
68          PUBLIC ${LUA_DEFINITIONS}
69      )
70      target_compile_options(lua_static
71          PRIVATE "-Wall" "-Wextra" "-fPIC"
72      )
73  endif()
74
75  if(LUA_BUILD_BINARY)
76      include(CheckIncludeFile)
77      CHECK_INCLUDE_FILE("readline/readline.h" HAVE_READLINE_READLINE_H)
78  endif()
```

Рисунок 4.2 – Флаг для сборки проекта обратной интеграции в .so

Сама сборка проекта производится так же, как и при интеграции Lua в C++:

```
practices/cCrossCross2 (cd build; cmake ..; cmake --build .)
CMake Deprecation Warning at CMakeLists.txt:1 (cmake_minimum_required):
  Compatibility with CMake < 3.10 will be removed from a future version of
  CMake.

  Update the VERSION argument <min> value. Or, use the <min>...<max> syntax
  to tell CMake that the project requires at least <min> but has been updated
  to work with policies introduced by <max> or earlier.

CMake Deprecation Warning at lua/CMakeLists.txt:1 (cmake_minimum_required):
  Compatibility with CMake < 3.10 will be removed from a future version of
  CMake.

  Update the VERSION argument <min> value. Or, use the <min>...<max> syntax
  to tell CMake that the project requires at least <min> but has been updated
  to work with policies introduced by <max> or earlier.

-- Configuring done (0.0s)
-- Generating done (0.0s)
-- Build files have been written to: practices/cCrossCross2/build
ninja: no work to do.
```

Рисунок 4.3 – Сборка проекта обратной интеграции

4.2 Тестирование

Тестовый код будет делать то же самое, что и при обычной интеграции: вызывать функцию нахождения простых чисел и засекаеть время выполнения:

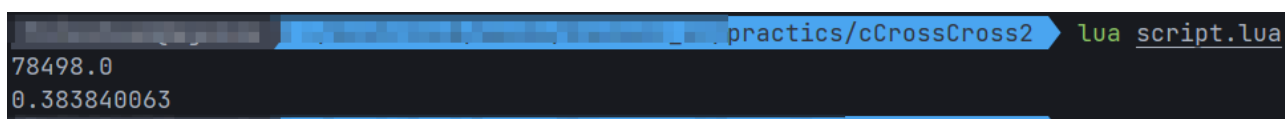
```
1  -- подключение библиотеки
2  rut = require("librut")
3
4  -- обращение происходит через .
5  timer = rut.TimerStart()
6  print(rut.Primes(1000000))
7  print(rut.TimerEnd(timer))
```

Рисунок 4.4 – Код скрипта Lua

Теперь исполняемой средой будет непосредственно runtime Lua, предварительно установленный в системе. Запуск происходит следующей командой:

```
lua script.lua
```

В результате, получается следующий вывод в консоли:



```
practices/cCrossCross2 lua script.lua
78498.0
0.383840063
```

Рисунок 4.5 – Результат запуска собственной библиотеки для Lua

Как видим, простые числа были успешно найдены, за то же время, что и в главе 6.

ВЫВОД

В ходе выполнения индивидуального задания была проведена двухсторонняя интеграция языков программирования Lua и C++ с использованием IDE VSCodium и кроссплатформенного инструмента сборки CMake. Разработан программный код, который демонстрирует успешную работу интеграции. В процессе работы над проектом были выполнены следующие задачи:

1. **Изучение и применение принципов интеграции Lua с C++.** Были изучены основные подходы к взаимодействию между Lua и C++ с использованием библиотеки Lua. Созданы примеры кода, демонстрирующие вызов Lua-скриптов из C++ и управление C++ объектами в Lua.

2. **Настройка рабочей среды и инструментов сборки.** Для разработки был использован редактор VSCodium, который обеспечил удобное и эффективное написание кода. Были настроены инструменты сборки CMake, что позволило автоматизировать процесс компиляции и управления зависимостями между библиотеками Lua и C++.

3. **Создание и тестирование интеграции Lua и C++.** Реализована двухсторонняя интеграция, при которой C++ может вызывать Lua-функции, а Lua может работать с C++ объектами. Это дало возможность реализовать гибкое взаимодействие между языками программирования, где C++ используется для высокой производительности и работы с системными ресурсами, а Lua – для написания высокоуровневой логики работы приложения.

4. **Защита и обоснование применения.** Для защиты отчета был подготовлен доклад, обосновывающий необходимость двухсторонней интеграции Lua и C++ в разработке программного обеспечения, а также ее применение в реальных задачах, требующих гибкости в программировании и высокой производительности.

Результаты выполнения задания продемонстрировали успешное использование двух языков программирования для решения задач, требующих как гибкости скриптов, так и производительности системного программирования.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Методологии разработки программного продукта / Testengineer – [Электронный ресурс] URL: <https://testengineer.ru/development-methodologies-sys-anal/> (дата обращения: 18.12.2024)
2. 8 лучших методологий разработки ПО в 2024 году / Purrweb – [Электронный ресурс] URL: <https://www.purrweb.com/ru/blog/metodologii-dlya-razrabotki-po/> (дата обращения: 18.12.2024)
3. CMake и C++ — братья навек / Хабр – [Электронный ресурс] URL: <https://habr.com/ru/articles/461817/> (дата обращения: 18.12.2024)
4. CMake documentation and community – [Электронный ресурс] URL: <https://cmake.org/documentation/> (дата обращения: 18.12.2024)
5. Что такое скрипты и с чем их едят – [Электронный ресурс] URL: <https://habr.com/ru/articles/196272/> (дата обращения: 18.12.2024)
6. LuaBridge 3.0 Reference Manual – [Электронный ресурс] URL: <https://kunitoki.github.io/LuaBridge3/Manual> (дата обращения: 18.12.2024)
7. Integrating Lua in C++ - GeekForGeeks – [Электронный ресурс] URL: <https://www.geeksforgeeks.org/integrating-lua-in-cpp/> (дата обращения: 18.12.2024)
8. foresteamnd - Super useful library for C++ - GitHub – [Электронный ресурс] URL: <https://github.com/Foresteam/foresteamnd> (дата обращения: 18.12.2024)
9. CMake-based build of Lua (5.4.6 and 5.3.3) - GitHub – [Электронный ресурс] URL: <https://github.com/walterschell/Lua> (дата обращения: 18.12.2024)

ПРИЛОЖЕНИЯ

Файл проекта NodeJS – package.json

```
1 {  
2   "name": "lua",  
3   "version": "1.0.0",  
4   "main": "index.js",  
5   "type": "module",  
6   "license": "MIT",  
7   "dependencies": {  
8     "js-crypto-aes": "^1.0.4"  
9   }  
10 }
```

Файл ключа – key.js

```
1 export const salt = new Uint8Array([
2   ...
3 ]);
4 export const key = new Uint8Array([
5   ...
6 ]);
7 export const iv = new Uint8Array([
8   ...
9 ]);
```

Основной исполняемый файл – compile.js

```
1 import fs from 'fs';
2 import aes from 'js-crypto-aes';
3 // импортируем заранее сгенерированный ключ
4 import { salt, key, iv } from './key.js';
5
6 const bytesToString = (bytes) =>
7   Array.from(bytes)
8     .map((b) => '\\x' + b.toString(16).padStart(2, '0'))
9     .join('');
10 const bytesToArray = (bytes) =>
11   '{ ' +
12   Array.from(bytes)
13     .map((b) => '0x' + b.toString(16).padStart(2, '0'))
14     .join(', ') +
15   ' }';
16
17 fs.readdirSync('.')
18   .filter((v) => v.endsWith('.lua'))
19   .forEach(async (v) => {
20     v = v.substring(0, v.length - 4);
21     // преобразование скрипта в C ++ строку
22     let content = fs
23       .readFileSync(`${v}.lua`, { encoding: 'utf-8' })
24       .toString('utf-8')
25       .split('\\n')
26       .join('\\\\n')
27       .split('\\n')
28       .map((v) => v.trim())
29       .filter((v) => v.length > 0 && !v.startsWith('--'))
30       .join('\\n')
31       .replace(/\\{\\n/g, '{ ')
32       .replace(/,\\n/g, ', ')
33       .replace(/\\n\\}/g, ' }');
34     // .replace(/"/g, '\\\\"').replace(/\\'/g, '\\\\\'')
35     // .split('\\n')
36     // .join('\\n');
```

```

37
38     // шифрование в байтах
39     content = await aes.encrypt(new TextEncoder().encode(content),
40         key, {
41             name: 'AES-CBC',
42             iv,
43         });
44     const size = content.byteLength;
45     // получение строки из массива байтов
46     content = bytesToString(content);
47
48     // запись заголовочный файл C ++ полученного скрипта
49     await fs.writeFileSync(
50         `${v}.h`,
51         `#pragma once
52 const char* script_${v} = "${content}";
53 const unsigned long int script_${v}_len = ${size};`,
54     );
55
56 // Запись ключа AES в файл C++
57 fs.writeFileSync(
58     'Key.h',
59     `#pragma once
60 namespace AES {
61     const unsigned char salt[] = ${bytesToArray(salt)};
62     const unsigned long int salt_size = ${salt.byteLength};
63     const unsigned char key[] = ${bytesToArray(key)};
64     const unsigned long int key_size = ${key.byteLength};
65     const unsigned char iv[16] = ${bytesToArray(iv)};
66 }`,
67 );

```

Файл проекта – CMakeLists.txt

```
1 cmake_minimum_required(VERSION 3.2.0)
2 project(rut VERSION 0.1.0)
3
4 add_subdirectory(lua)
5 add_custom_target(luacompile ALL
6     COMMAND node compile.js
7     WORKING_DIRECTORY ../src/lua
8     SOURCES ../src/lua/Startup.lua ../src/lua/compile.js
9     BYPRODUCTS ../src/lua/Startup.h ../src/lua/Key.h
10 )
11
12 include(CTest)
13 enable_testing()
14
15 add_executable(rut
16     src/lua/Startup.h
17     src/LuaFunctions.h
18     src/LuaFunctions.cpp
19     src/main.cpp
20 )
21 add_dependencies(rut luacompile)
22
23 target_include_directories(rut
24     PUBLIC include
25     PUBLIC include/LuaBridge
26     PUBLIC lua/lua-5.4.4/include
27 )
28
29 if(UNIX AND NOT APPLE)
30 elseif(WIN32)
31 endif()
32
33 target_link_libraries(rut
34     foresteamnd
35     lua_static
36     ${PLATFORM_LIBS})
```

```
37 | )  
38 | set (CPACK_PROJECT_NAME ${PROJECT_NAME})  
39 | set (CPACK_PROJECT_VERSION ${PROJECT_VERSION})  
40 | include (CPack)
```

Основной файл проекта C++ – main.cpp

```
1 #include "LuaFunctions.h"
2 #include "lua/Key.h"
3 #include "lua/Startup.h"
4 #include <foresteamnd/Utils>
5 #include <fstream>
6 #include <iostream>
7 #include <list>
8 #include <plusaes/plusaes.hpp>
9 #include <string.h>
10 #include <thread>
11
12 // AES дешифрование
13 void Decrypt(const char* data, size_t length, char* output) {
14     size_t padded_size;
15     plusaes::decrypt_cbc((const unsigned char*)data, length, AES::key
16         , AES::key_size, &AES::iv, (unsigned char*)output, length, &
17         padded_size);
18 }
19
20 int LuaPrimes(lua_State* L, int number) {
21     // получаем функцию
22     lua_getglobal(L, "LuaPrimes");
23     // помещаем аргумент в стек
24     lua_pushnumber(L, number);
25     // вызываем функцию
26     lua_pcall(L, 1, 1, 0);
27     double result = lua_tonumber(L, -1);
28     // очищаем стек
29     lua_pop(L, 1);
30
31     return result;
32 }
33
34 string RemoteExecReadline(lua_State* L, string command) {
35     lua_getglobal(L, "RemoteExec");
36     lua_pushstring(L, command.c_str());
```

```

35     lua_pcall(L, 1, 1, 0);
36     string result = "empty string";
37     if (lua_isstring(L, -1))
38         result = lua_tostring(L, -1);
39     lua_pop(L, 1);
40
41     return result;
42 }
43
44 int main(int, char**) {
45     lua_State* L;
46
47     // расшифровываем скрипт в память
48     char* dStartup = new char[script_Startup_len]();
49     Decrypt(script_Startup, script_Startup_len, dStartup);
50
51     // инициализируем Lua
52     L = luaL_newstate();
53
54     // регистрируем библиотеки
55     luaL_openlibs(L);
56     // в том числе собственные
57     LuaFunctions::Register(L);
58
59     // выполняем скрипт
60     luaL_dostring(L, dStartup);
61
62     // // достаем Lua функцию - в C++
63     // lua_getglobal(L, "IsWindows");
64     // // вызываем функцию
65     // lua_pcall(L, 0, 1, 0);
66     // cout << "Am i on windows? " << (lua_toboolean(L, -1) ? "true"
        // : "false") << endl;
67     // lua_pop(L, 1);
68
69     // // вызываем другую функцию , но уже с аргументом
70     // auto t = LuaFunctions::TimerStart();

```



```

71 // cout << "LuaPrimes from C++: " << LuaPrimes(L, 1000000) <<
    endl;
72 // cout << "Elapsed: " << LuaFunctions::TimerEnd(t) << " sec" <<
    endl;
73
74 cout << "Enter a script to execute: ";
75 string command;
76 getline(cin, command);
77 string output = RemoteExecReadline(L, command);
78 cout << "Output: " << output << endl;
79
80 // очистка
81 lua_close(L);
82
83 delete[] dStartup;
84 return 0;
85 }

```

Заголовочный файл экспортируемых в Lua функций – LuaFunctions.h

```
1 #pragma once
2 extern "C" {
3 #include <lauxlib.h>
4 #include <lua.h>
5 #include <lualib.h>
6 }
7 #include <LuaBridge.h>
8 #include <string>
9 using namespace std;
10 using namespace luabridge;
11
12 namespace LuaFunctions {
13     void Register(lua_State* L);
14     int64_t TimerStart();
15     double TimerEnd(int64_t timer);
16 };
```

Файл исходного кода экспортируемых в Lua функций – LuaFunctions.cpp

```
1 #include "LuaFunctions.h"
2 #include <array>
3 #include <chrono>
4 #include <foresteamnd/Utils>
5 #include <fstream>
6 #include <memory>
7 #include <sstream>
8 #include <stdexcept>
9 #include <thread>
10
11 std::string exec(string cmd) {
12     std::array<char, 128> buffer;
13     std::string result;
14     std::unique_ptr<FILE, decltype(&pclose)> pipe(popen(cmd.c_str(),
15         "r"), pclose);
16     if (!pipe) {
17         throw std::runtime_error("popen() failed!");
18     }
19     while (fgets(buffer.data(), buffer.size(), pipe.get()) != nullptr
20         ) {
21         result += buffer.data();
22     }
23     return result;
24 }
25
26 auto timers = vector<chrono::_V2::system_clock::time_point>();
27 namespace LuaFunctions {
28     int64_t TimerStart() {
29         auto start = chrono::high_resolution_clock::now();
30         int64_t i = timers.size();
31         timers.push_back(start);
32         return i;
33     }
34     double TimerEnd(int64_t timer) {
35         auto end = chrono::high_resolution_clock::now();
```

```

34     chrono::duration<double, std::milli> duration = end - timers[
        timer];
35     return duration.count() / 1000.;
36 }
37
38 /// @param msg Message
39 /// @param title Title
40 void lAlert(const string& msg, const string& title = "") {
41     system(("kdialog --msgbox \"" + msg + "\" --title \"" + title +
        "\"").c_str());
42 }
43
44 /// @param cmd Command
45 /// @returns Result of execution
46 string lExec(const string& cmd) {
47     return exec(cmd);
48 }
49
50 void lSleep(const int64_t ms) {
51     std::this_thread::sleep_for(std::chrono::milliseconds(ms));
52 }
53
54 int64_t lCFactorial(int64_t n) {
55     if (n < 0) {
56         std::cerr << "Error: Factorial of a negative number is
            undefined.\n";
57         return 0; // Return 0 for invalid input
58     }
59
60     int64_t result = 1;
61     for (int i = 1; i <= n; ++i)
62         result *= i;
63
64     return result;
65 }
66
67 bool IsPrime(int64_t n) {

```

```

68     if (n < 2)
69         return false;
70     for (int i = 2; i <= sqrt(n); i++)
71         if (n % i == 0)
72             return false;
73     return true;
74 }
75 int64_t lCPrimes(int64_t n) {
76     int64_t primes = 0;
77     for (int i = 2; i <= n; i++)
78         if (IsPrime(i))
79             primes = primes + 1;
80     return primes;
81 }
82
83 void Register(lua_State* L) {
84     getGlobalNamespace(L)
85         .addFunction("_Exec", lExec)
86         .addFunction("Sleep", lSleep)
87         .addFunction("CFactorial", lCFactorial)
88         .addFunction("TimerStart", TimerStart)
89         .addFunction("TimerEnd", TimerEnd)
90         .addFunction("CPrimes", lCPrimes)
91
92         .beginNamespace("test")
93         .addFunction("Alert", lAlert)
94         .endNamespace();
95 }
96 };

```

Скрипт Lua – Startup.lua

```
1 string.split = function(inputstr, sep)
2   if sep == nil then
3     sep = "%s"
4   end
5   local t = {}
6   for str in string.gmatch(inputstr, "([^\s".. sep .. "\s]+)") do
7     table.insert(t, str)
8   end
9   return t
10 end
11
12 function FileExists(filename)
13   local f = io.open(filename, "r")
14   if f then f:close() end
15   return f ~= nil
16 end
17
18 function Exec(command)
19   print(_Exec(command))
20 end
21
22 function IsWindows()
23   return string.sub(os.getenv('HOME'), 1, 1) ~= '/'
24 end
25
26 function IsPrime(n)
27   if n < 2 then
28     return false
29   end
30   for i = 2, math.sqrt(n) do
31     if n % i == 0 then
32       return false
33     end
34   end
35   return true
36 end
```

```

37 function LuaPrimes(n)
38     local primes = 0
39     for i = 2, n do
40         if IsPrime(i) then
41             primes = primes + 1
42         end
43     end
44     return primes
45 end
46
47 function RemoteExec(code)
48     return load(code, 'rExec')()
49 end
50
51 -- Exec("neofetch")
52 -- test.Alert("Don't tell me i don't need this anymore", "Don't
    tell me you don't feel the same inside")
53 -- print(IsWindows() and "on windows" or "on unix")
54
55 -- t = TimerStart()
56 -- print("CPrimes from Lua:", CPrimes(1000000))
57 -- print("Elapsed", TimerEnd(t), "sec")

```

Файл проекта обратной интеграции – CMakeLists.txt

```

1  cmake_minimum_required(VERSION 3.2.0)
2  project(rut VERSION 0.1.0)
3
4  add_subdirectory(lua)
5
6  include(CTest)
7  enable_testing()
8
9  add_library(rut SHARED
10     src/Timer.h
11     src/Primes.h
12     src/Timer.cpp
13     src/Primes.cpp

```

```

14     src/main.cpp
15 )
16
17 target_include_directories(rut
18     PUBLIC lua/lua-5.4.4/include
19 )
20
21 target_link_libraries(rut
22     foresteamnd
23     lua_static
24     ${PLATFORM_LIBS}
25 )
26 set(CPACK_PROJECT_NAME ${PROJECT_NAME})
27 set(CPACK_PROJECT_VERSION ${PROJECT_VERSION})
28 include(CPack)

```

Основной файл проекта обратной интеграции C++ – main.cpp

```

1 #include <chrono>
2 #include <foresteamnd/Utils>
3 #include <fstream>
4 #include <iostream>
5 #include <list>
6 #include <string.h>
7 #include <thread>
8
9 extern "C" {
10 #include <lauxlib.h>
11 #include <lua.h>
12 #include <lualib.h>
13 }
14
15 #include "Primes.h"
16 #include "Timer.h"
17
18 static int lTimerStart(lua_State* L) {
19     lua_pushinteger(L, TimerStart());
20     return 1;

```



```

21 }
22 static int lTimerEnd(lua_State* L) {
23     int64_t timer = luaL_checkinteger(L, -1);
24     lua_pushnumber(L, TimerEnd(timer));
25     return 1;
26 }
27
28 static int lPrimes(lua_State* L) {
29     int64_t n = luaL_checkinteger(L, -1);
30     lua_pushnumber(L, Primes(n));
31     return 1;
32 }
33
34 // Library open function
35 extern "C" int luaopen_librut(lua_State* L) {
36     static const luaL_Reg librut[] = {
37         {"TimerStart", lTimerStart},
38         {"TimerEnd", lTimerEnd},
39         {"Primes", lPrimes},
40         {nullptr, nullptr} // Sentinel item
41     };
42
43     luaL_newlib(L, librut);
44     return 1;
45 }

```