Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 23.Б10-мм

C-совместимость операционной системы, реализованной на rust

Нестеренко Артём Дмитриевич

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
ст. преподаватель кафедры ИАС, К.К. Смирнов

Консультант:
Ведущий разработчик ПО «Софтком», И.С. Архипов

Санкт-Петербург
2025

Введение

Для правильной работы компьютера необходима операционная система. Она управляет памятью, ресурсами, вводом и выводом данных, выполняет манипуляции с файлами, управляет внешними устройствами. В мире существует множество операционных систем, разработка новых ведется и по сей день.

В компании «Софтком» ведется разработка операционной системы Martos, которая представляет собой прототип операционной системы реального времени. Martos написана на языке Rust, но прикладные программы можно писать на C. Martos поддерживает архитектуры процессоров mips64, risc-v-esp32-c6, xtensa-esp32.

Данную операционную систему планируется использовать как замену операционной системы, которая используется в компании на данный момент ввиду тяжеловесности и отсутствия открытого исходного кода последней.

Однако для написания программного обеспечения на языке C для архитектуры процессора xtensa-esp32 необходимо разрешить множество задач связанных с реализацией подобного программного обеспечения. Одной из главных проблем является написание и сборка библиотек операционной системы. Также необходим программный интерфейс, который можно представить в виде отдельного модуля, ведь последний позволил бы осуществлять правильное взаимодействие операционной системы и программ на определенном языке.

Создание программного интерфейса для языка C в виде отдельного модуля для архитектур процессоров mips64, risc-v-esp32-c6, xtensa-esp32, а также исправление библиотек операционной системы позволит писать прикладные программы для Martos на соответствующем языке и соответствующих архитектурах.

Дата компиляции: 18.10.2024

1. Постановка задачи

Целью работы является осуществление возможности написания прикладных программ на C для операционной системы Martos на базе архитектуры xtensa, а также представление API для языка C в виде отдельного модуля. Для её выполнения были поставлены следующие задачи:

1. Доработать имеющийся API на языке C;
2. Доработать пример, демонстрирующий работоспособность API для архитектуры xtensa;
3. Проверить работоспособность на демонстрационном примере.

2. Обзор

Для написания операционных систем язык C является одним из наиболее частых выборов среди разработчиков. На нем написаны самые популярные операционные системы: Windows, Linux, macOS. Язык Rust появился намного позже C, но он имеет возможности для реализации на нем операционных систем. На нем, например, написана Redox OS. Операционная система Martos написана на Rust, но также нужно реализовать и поддержку C для написания программных приложений. Разработка Martos была мотивирована желанием использовать операционную систему, которая будет удовлетворять основным потребностям, но не будет иметь ничего лишнего (в рамках нужд компании).

2.1. Обзор существующих решений

Martos имеет программный интерфейс, но лишь для одной из поддерживаемых операционной системой архитектуры процессора — esp32 xtensa. Данный программный интерфейс платформонезависимый, поэтому его можно портировать и на другие архитектуры процессоров, такие как mips64 и risc-v-esp32-c6. Необходимо вынести программный интерфейс в отдельный модуль, чтобы его можно было использовать для вышеперечисленных архитектур процессоров. Также для демонстрации совместимости операционной системы с языком C существует пример, написанный на языке C, собирающийся и запускающийся на плате esp32. Существует проблема, что при вызове функции, отвечающей за инициализацию операционной системы, сборка перестает собираться и выдает ошибку, предположительно из-за того, что мы дважды линкуем одни и те же библиотеки Martos. Данную проблему необходимо разрешить.

2.2. Обзор используемых технологий

- RustRover[2]

RustRover — интегрированная среда разработки для программирования на Rust, созданная компанией JetBrains.

- C[3]

  C — низкоуровневый компилируемый язык программирования со статической типизацией, разработанный в 1972.

- Rust[4]

  Rust — низкоуровневый компилируемый язык программирования со статической типизацией, являющийся мультипарадигменным.

- Martos[1]

  Martos — операционная система, предоставляющая возможность разработчикам писать программное обеспечение на языках Rust и C.

2.3. Выводы

Martos уже имеет наработки для совместимости с языком C, но эти наработки необходимо довести до готового состояния: реализовать уже готовый программный интерфейс для архитектуры xtensa в виде отдельного модуля, который можно использовать для архитектур процессоров mips64, risc-v-esp32-c6, а также устранить ошибку в демонстрационном примере на C, чтобы в нем можно было инициализировать систему.

3. Описание решения

Пример на C собирался с использованием утилиты make версии 4.4.1. Для работы с API для языка C использовалась система сборки Cargo версии 1.81.0. Для разработки под esp32 использовался SDK ESP-IDF версии 5.2. Использовался xtensa-esp32-elf-gcc — это кросс-компилятор для процессоров Xtensa, который является частью ESP-IDF. Использовался gcc версии 14.1.0.

3.1. Реализация модуля программного интерфейса для C

У Martos имеются функции, которые являются общими для всех поддерживаемых архитектур. Среди них:

- «init_system» для инициализации Martos;
- «setup_timer» для настройки параметров конфигурации таймера;
- «add_task» для добавления задач в менеджер задач;
- «start_task_manager» для запуска менеджера задач.

При применении Cargo создается проект, состоящий из:

- Cargo.toml — манифест с метаданными проекта, содержащий название проекта, зависимости и т.д.;
- Cargo.lock — файл с подробной информацией о зависимостях проекта;
- src/lib.rs — файл с исходным кодом, предназначен для сборки библиотеки, а не исполняемой программы.

В директории src также находятся task_manager.rs и timer.rs. Эти файлы вместе с lib.rs содержат все вышеперечисленные и нужные функции.

В отдельный созданный файл src/c_api.rs добавляются функции, которые можно будет вызывать из кода на C. Они будут ссылаться

на нужные функции на Rust. Чтобы это сделать, предпринимаются следующие шаги:

- использование для функций аттрибута #[no_mangle], который сообщает компилятору, что имя функции не должно модифицироваться при компиляции. Rust меняет имена функций, чтобы гарантировать уникальность имён между различными библиотеками, но с данным аттрибутом таких изменений в именах не будет;
- объявление функций с помощью конструкции «extern "C"». Это нужно для указания того, что функция придерживается соглашения о вызове из C.

Указывается ссылка на lib.rs проекта Martos в директории c-library, в которой содержатся три Cargo проекта для xtensa-esp32, risc-v-esp32-c6, mips64, которые тоже собираются в виде библиотек, как и основной проект. Предварительно добавляется ссылка на c_api.rs в src/lib.rs.

3.2. Исправление примера на C для xtensa-esp32

Имеется пример, находящийся в том же репозитории в директории examples/c-examples/xtensa-esp32, демонстрирующий совместимость операционной системы с языком C для xtensa-esp32. Он состоит из:

- main.c — файл с исходным кодом примера;
- Makefile — файл для указания правил сборки программы, в частности указаний для линковки со статической библиотекой, содержащей функции из вышеупомянутого программного интерфейса;
- esp32.ld — скрипт для распределения памяти примера.

При добавлении в main.c функции для инициализации операционной системы возникают ошибки, которые сигнализируют о нехватке выделенной памяти для хранения переменных и данных, а также о неопределенных ссылках на некоторые функции из пакета esp-hal.

Проблема нехватки выделенной памяти говорит о том, что область памяти «dram_seg», указанная в esp32.ld, переполнена. Нынешняя и старая карты памяти платы esp32 имеют 520 килобайт для хранения данных и инструкций, разделяя эту память на секции IRAM для хранения исполняемого кода и DRAM для хранения переменных, стека, кучи. В отличии от старой версии карты памяти, в нынешней версии имеется поддержка PSRAM — это внешняя память, которая подключается к esp32 и используется для расширения объема оперативной памяти. Оперативная память карты памяти esp32 делится между двумя областями в esp32.ld, а именно между «iram_seg» и «dram_seg». Эти области используются для хранения кода и данных соответственно, области имеют свой адрес, размер выделенной им памяти. Для «iram_seg» выделено 63 килобайта, а для «dram_seg» 4 килобайта. В ошибке сказано, что область переполнена чуть больше чем на 60 килобайт. Увеличив размер выданной памяти на это число, что возможно, так как не происходит выхода за рамки имеющейся у платы памяти, ошибка будет устранена.

В основном репозитории имеется директория esp-idf, в ней содержатся библиотеки и инструменты для работы с esp32. Функции из-за которых появляется ошибка о неопределенных ссылках — это «ets_delay_us», «rom_i2c_writeReg». Обе эти функции объявлены внутри заголовочных файлов, которые находятся внутри директории esp-idf/components/hal/esp32/include. Ошибка возникает при попытке использовать функции из esp-hal, которые вызывают одну из двух вышеупомянутых функции. Внутри той же директории находится файл esp-idf/components/esp_rom/esp32/ld/esp32.rom.ld, в котором указано, по каким адресам находится та или иная функция. Функции включаются по их адресам в нужный esp32.ld, чтобы указать компоновщику на их расположение, тогда ошибка будет устранена.

4. Эксперимент

Для проверки работоспособности примера на C небходима плата esp32 с процессором xtensa. Чтобы использовать в примере функции из Martos понадобится ранее описанный API для языка C. С его помощью будет собираться статическая библиотека с функциями Martos. Данную библиотеку необходимо включить в Makefile примера, чтобы библиотека линковалась с остальными составляющими примера. Предварительно устанавливаются инструменты для разработки под esp32 (python3, cmake, ninja-build, libusb, make, flex, bison), а также ESP-IDF.

Для сборки проекта необходимо настроить окружение с помощью команды . $HOME/esp/esp-idf/export.sh, которая сделает доступными инструменты ESP-IDF. Собирается проект с помощью команды make, которая выполнит инструкции из Makefile.

Для запуска проекта первоначально нужно загрузить программу, отформатировав ее для esp32. Делается это с помощью команды esptool.py –chip esp32 elf2image –flash_mode=”dio” –flash_freq ”40m” –flash_size ”4MB” -o main.bin main.elf

После сохранить ее в чипе SPI Flash, подключенном к реальному esp32 внутри модуля с помощью команды

esptool.py –chip esp32 –port /dev/ttyUSB0 –baud 115200 –before default_reset –after hard_reset write_flash -z –flash_mode dio –flash_freq 40m –flash_size detect 0x1000 main.bin

Запускается программа с помощью команды esptool.py –chip esp32 –port /dev/ttyUSB0 –baud 115200 –before default_reset –after hard_reset run Порт указывается в зависимости от того, к какому системному ресурсу подключен esp32.

В результате данная прошивка успешно отработает на контроллере esp32.

Заключение

По итогу работы были выполнены следующие задачи:

- Был доработан уже имеющийся API на языке C для xtensa-esp32, risc-v-esp32-c6, mips64, который используется для сборки статической библиотеки, содержащей функции из Martos. https://github.com/IvanArkhipov1999/Martos/pull/63;
- Был исправлен пример, демонстрирующий работоспособность API для xtensa-esp32. https://github.com/IvanArkhipov1999/Martos/pull/81 https://github.com/IvanArkhipov1999/Martos/pull/83;
- Проведены проверки на работоспособность примера и API.

Список литературы

[1] IvanArkhipov1999. Martos. — URL: https://github.com/IvanArkhipov1999/Martos (дата обращения: 18 октября 2024 г.).

[2] Wikipedia. RustRover. — URL: https://ru.wikipedia.org/wiki/RustRover (дата обращения: 18 октября 2024 г.).

[3] Wikipedia. Си (язык программирования). — URL: https://ru.wikipedia.org/wiki/Си_(язык_программирования) (дата обращения: 18 октября 2024 г.).

[4] Wikipedia. Rust (язык программирования). — 2024. — URL: https://ru.wikipedia.org/wiki/Rust_(язык_программирования) (дата обращения: 18 октября 2024 г.).