Санкт-Петербургский государственный университет

Кафедра системного программирования

Портирование операционной системы Martos под STM32F4

Максим Александрович Родионов, группа 23.Б15-мм

Научный руководитель: К. К. Смирнов, ст. преподаватель кафедры ИАС
Консультант: И. С. Архипов, ведущий разработчик ПО, ООО «СофтКом»

Санкт-Петербург
2025

Введение

- Операционные системы — основа взаимодействия между аппаратным обеспечением и приложениями
- Популярность ARM-архитектуры и доступность плат STM32
- Адаптация Martos под STM32F4 (ARM Cortex-M4)
- Библиотека `stm32f4xx-hal`

Постановка задачи

Целью данной работы является адаптация операционной системы Martos для работы на серии микроконтроллеров STM32F4

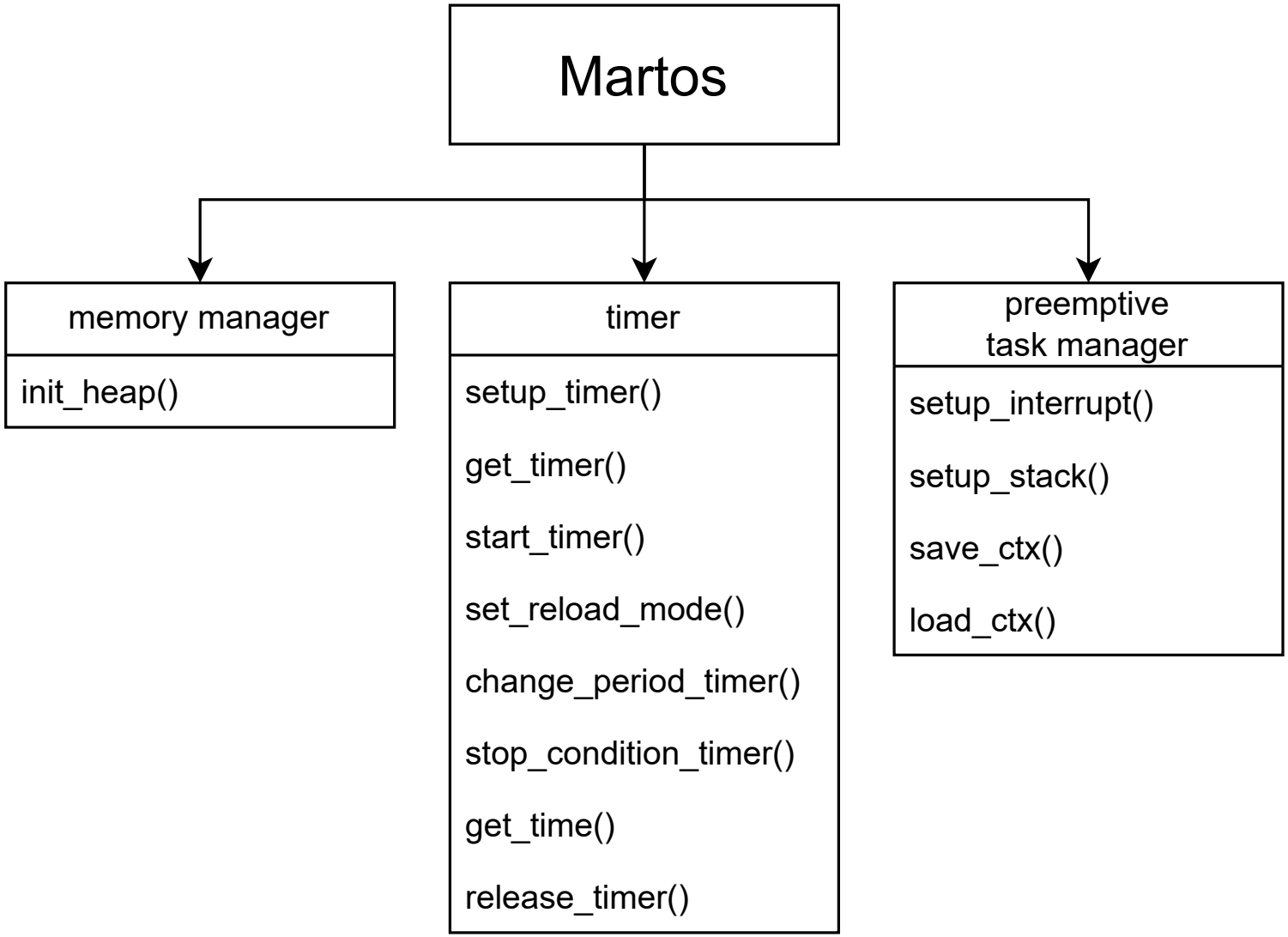
Задачи:

- Провести обзор основных компонентов операционной системы и доступных инструментов для разработки
- Реализовать поддержку базовых функций операционной системы, таких как управление динамической памятью, работа с таймерами, адаптация вытесняющего планировщика задач и поддержка UART[1]
- Разработать набор демонстрационных примеров на языках программирования Rust и C для проверки работоспособности основных компонентов Martos

[1] Universal Asynchronous Receiver-Transmitter

Обзор (1/2)

- Интерфейсы Martos:

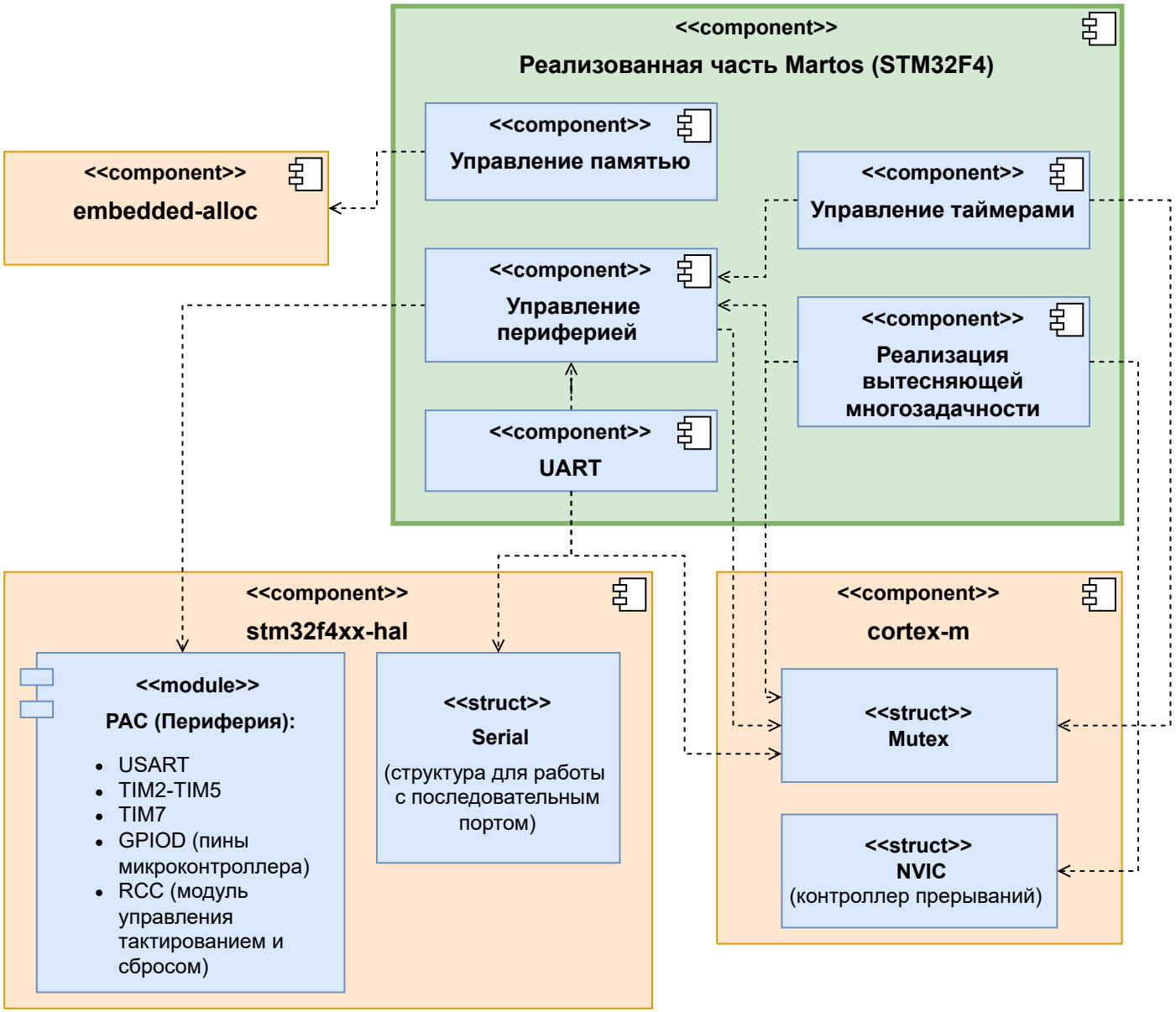


Обзор (2/2)

Инструменты для реализации:

- cortex-m
- stm32f4xx-hal
- embedded-alloc
- OpenOCD, GDB

Реализация


<<component>>
Реализованная часть Martos (STM32F4)
<<component>>
Управление памятью
<<component>>
Управление таймерами
<<component>>
Управление периферией
<<component>>
Реализация вытесняющей многозадачности
<<component>>
UART
<<component>>
embedded-alloc
<<component>>
stm32f4xx-hal
<<module>>
PAC (Периферия):
• USART
• TIM2-TIM5
• TIM7
• GPIOD (пины микроконтроллера)
• RCC (модуль управления тактированием и сбросом)
<<struct>>
Serial
(структура для работы с последовательным портом)
<<component>>
cortex-m
<<struct>>
Mutex
<<struct>>
NVIC
(контроллер прерываний)

Реализация: периферия, UART, управление памятью и таймерами

- Структура `Peripherals` для централизованного управления периферией
- Функция get_uart(), возвращающая настроенный экземпляр последовательного порта
- Библиотека `embedded-alloc` для управления памятью
- Реализация API аппаратных таймеров, разработанного в рамках предыдущего семестра[2]: setup_timer(), start_timer(), get_time() и другие

[2] https://se.math.spbu.ru/thesis/texts/Rodionov_Maksim_Aleksandrovich_Autumn_practice_2nd_year_2024_text.pdf

Реализация: вытесняющая многозадачность

- Настройка таймера для планирования задач
- Сохранение/восстановление контекста процессора:
  - Автоматически формирующийся стековый фрейм
  - Программное сохранение/восстановление регистров R4-R11, S16–S31, PSP[3]

[3] Process Stack Pointer

Примеры на Rust

1. hello-world:
   - Вывод сообщения через UART
2. dynamic-memory:
   - В рамках циклической задачи:
     - Увеличение значения глобального счётчика
     - Добавление текущего значение счетчика в вектор (VEC)
     - Вывод последнего элемента вектора
3. timer:
   - Настройка таймера
   - Запуск таймера и сохранение ссылки на него в глобальной переменной
   - Получение и вывод текущего значения таймера в рамках циклической задачи
   - Остановка таймера
4. scheduler:
   - Добавление двух задач в планировщик
   - Каждая задача увеличивает значение глобального счётчика на 1, выводит значение и имитирует выполнение длительной операции с помощью задержки

Пример на C

- Реализация инициализации системы
- Дерево объектов файловой системы примера:

```
c-example/
├── ld/
│   └── stm32f4.ld
├── src/
│   ├── startup.c
│   └── main.c
└── Makefile
```

Тестирование и апробация

- Работоспособность демонстрационных примеров проверялась на физической плате
- Интеграционные тесты в CI
- Апробация разработанного решения проводилась совместно с консультантом, являющимся также представителем компании
- Рассмотрены и одобрены:
  - Работоспособность
  - Степень соответствия реализованной функциональности предъявленным требованиям и ожиданиям

Результаты

- Проведен обзор основных компонентов операционной системы и доступных инструментов для разработки
- Реализована поддержка базовых функций операционной системы
- Разработан набор демонстрационных примеров на языках программирования Rust и C

Код работы является закрытым.

Выделение памяти

```rust
use core::mem::MaybeUninit;
use embedded_alloc::LlffHeap as Heap;

#[global_allocator]
static HEAP: Heap = Heap::empty();

/// Heap initialization.
pub fn init_heap() {
    const HEAP_SIZE: usize = 64 * 1024;
    static mut HEAP_MEM: [MaybeUninit<u8>; HEAP_SIZE] =
        [MaybeUninit::uninit(); HEAP_SIZE];
    unsafe {
        HEAP.init(HEAP_MEM.as_mut_ptr() as usize, HEAP_SIZE);
    }
}
```