Исключительная модель памяти

АЛЕКСЕЙ ТКАЧЕНКО
ОАО «ПЕЛЕНГ»
TKACHENKO@PELENG.BY, ALEXEY.TKACHENKO@GMAIL.COM
HTTPS://GITHUB.COM/ALEXEY-TKACHENKO/

Об авторе

Ведущий разработчик ОАО «Пеленг»

Специализируюсь на испытательном оборудовании для космической аппаратуры

В свободное время делаю pet-проекты, в том числе embedded

Вижу боль embedded-разработчиков и хочу помочь

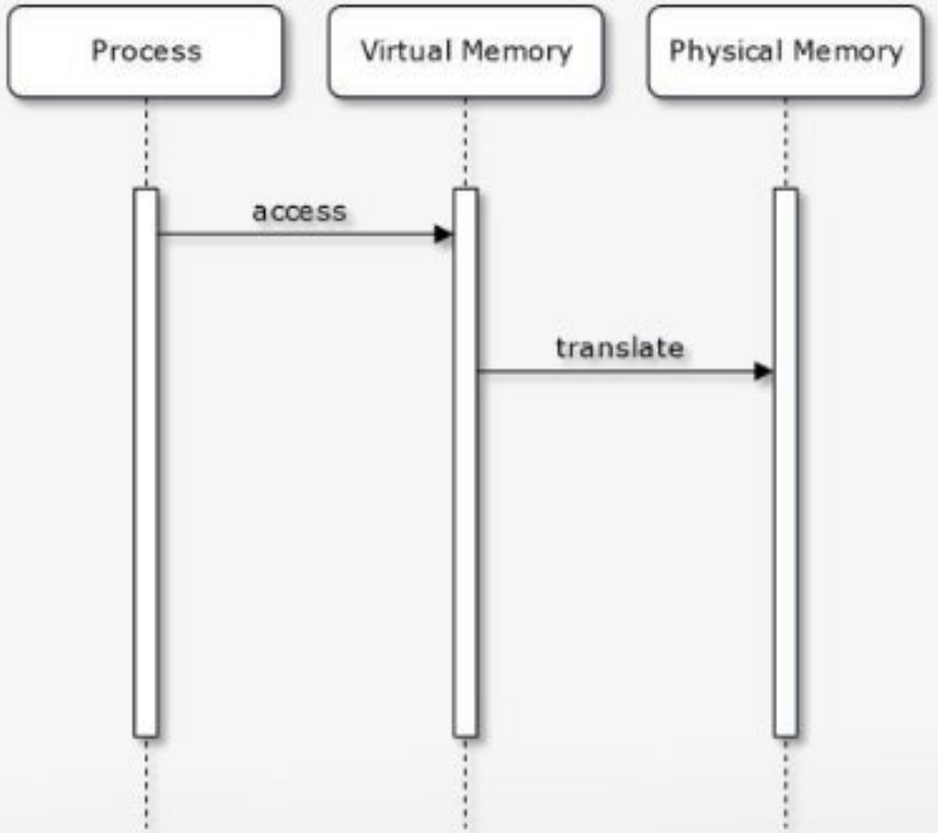
План доклада

Регистровые аппаратные интерфейсы

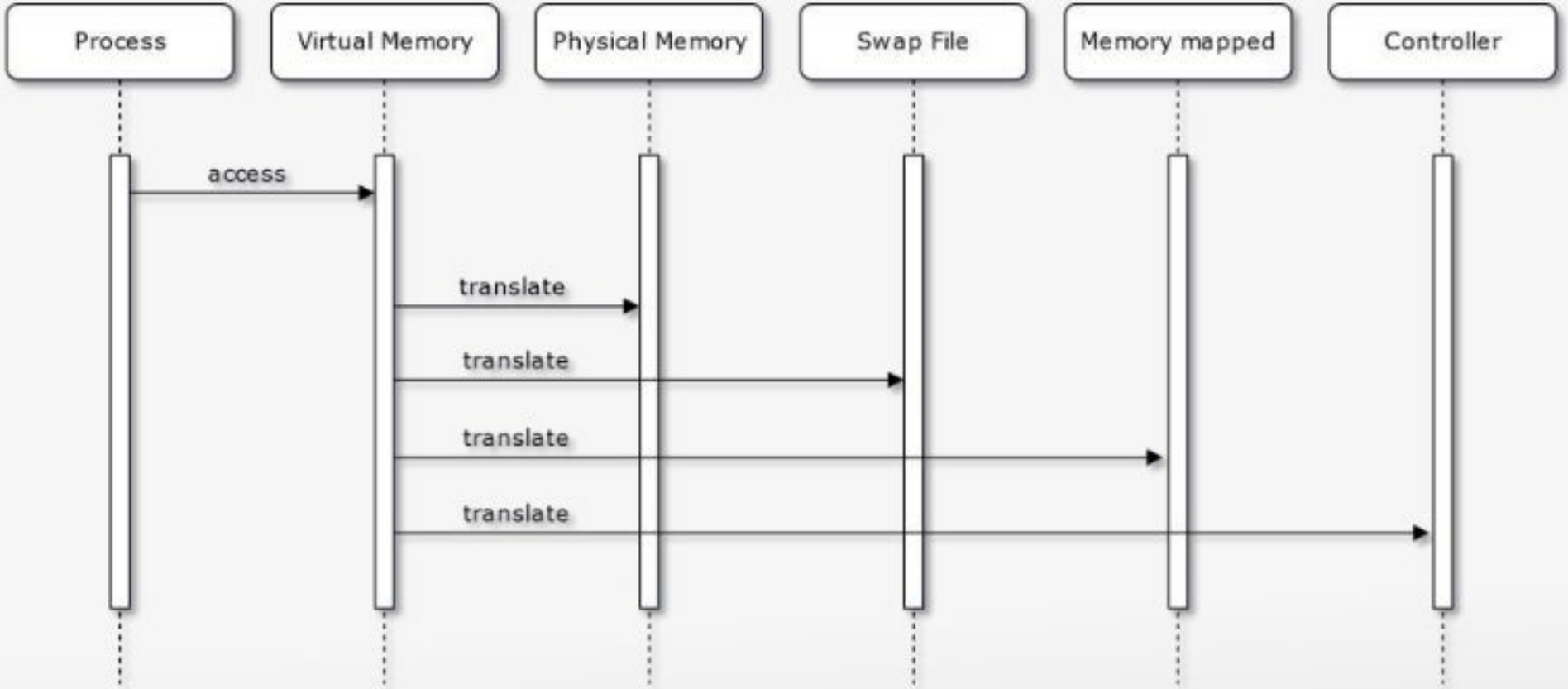
Имитация аппаратных интерфейсов с помощью механизмов защиты памяти

Библиотека ExMM для имитации регистрового аппаратного интерфейс

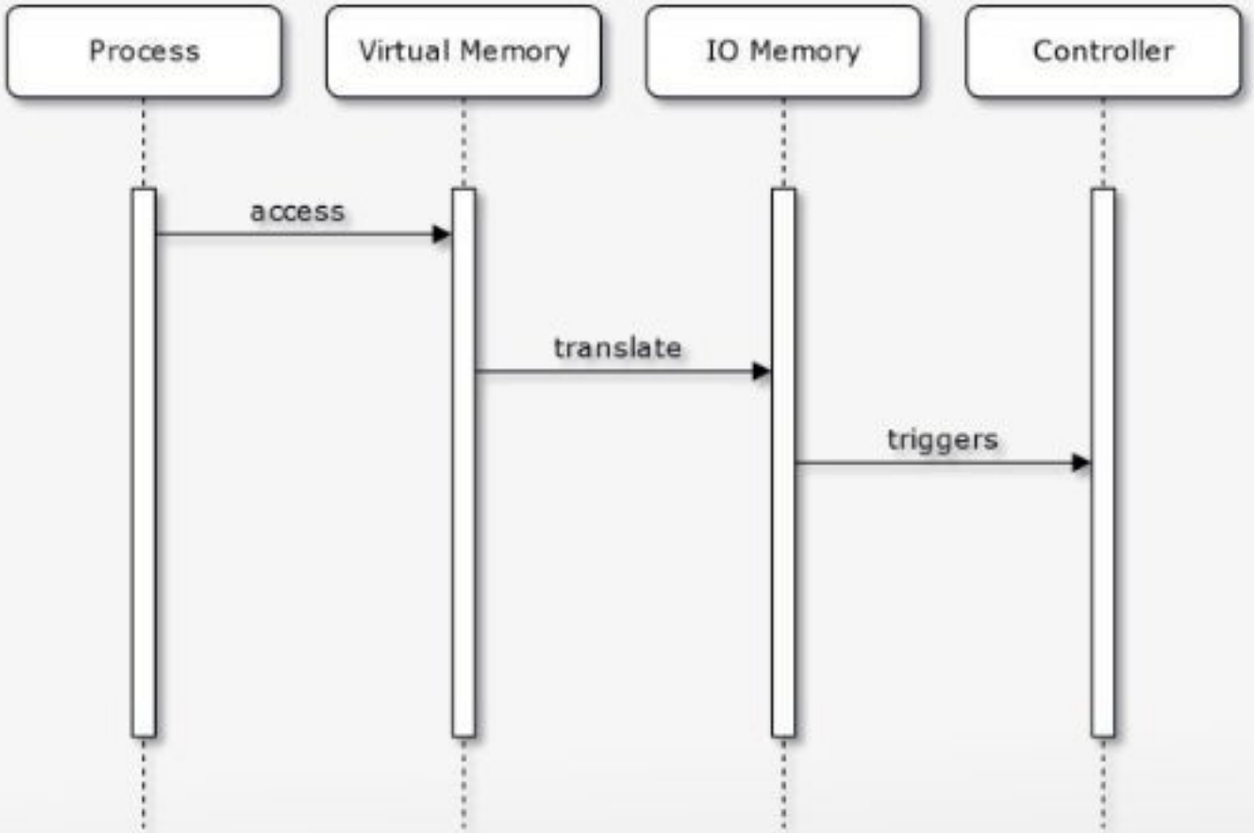
Трансляция адресов памяти



Трансляция адресов памяти



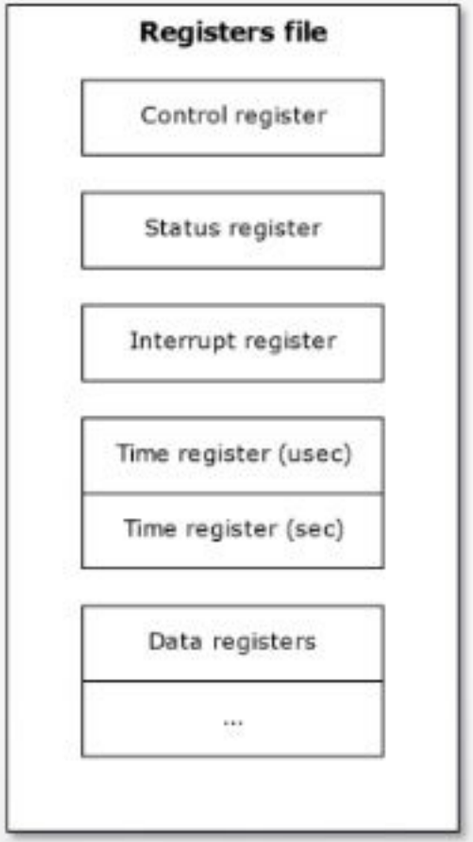
Модель ввода-вывода



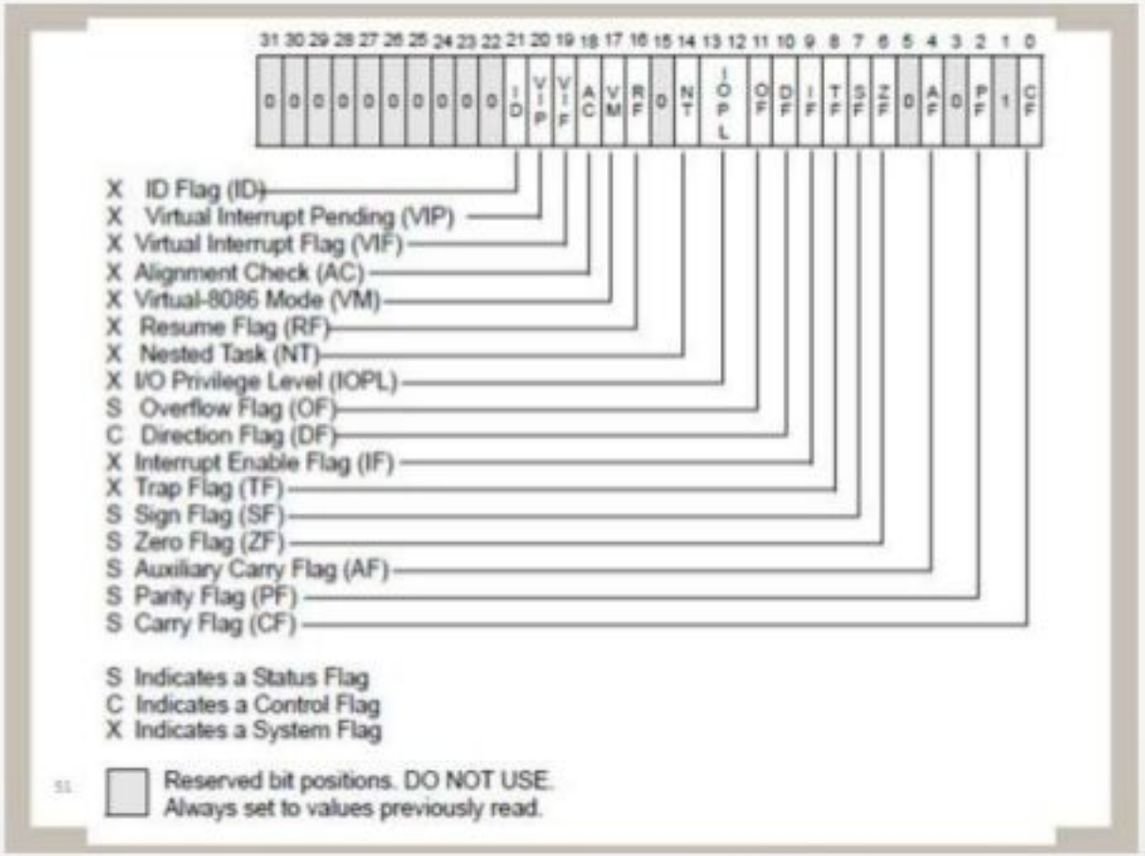
Модель ввода-вывода

Свойства регистров:

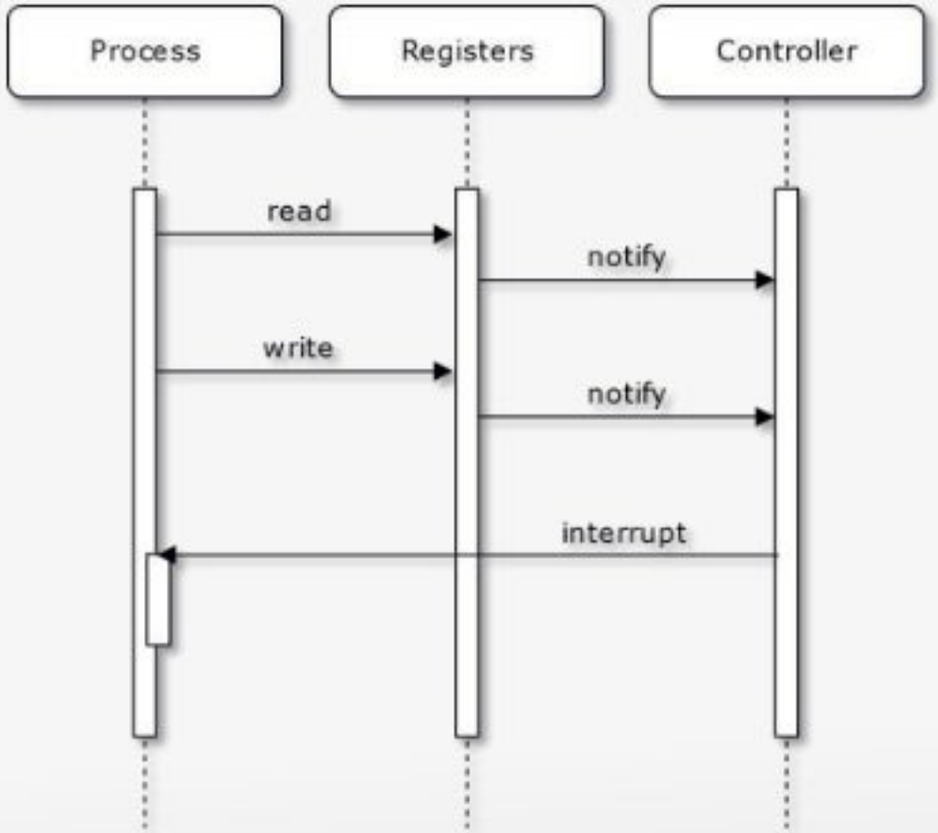
- Назначение битов определяется контроллером
- Биты не обязательно доступны для чтения/записи (пример – MSR, например EFlags в x86)
- Чтение/запись могут быть асимметричными
- В общем случае не предполагается хранение содержимого



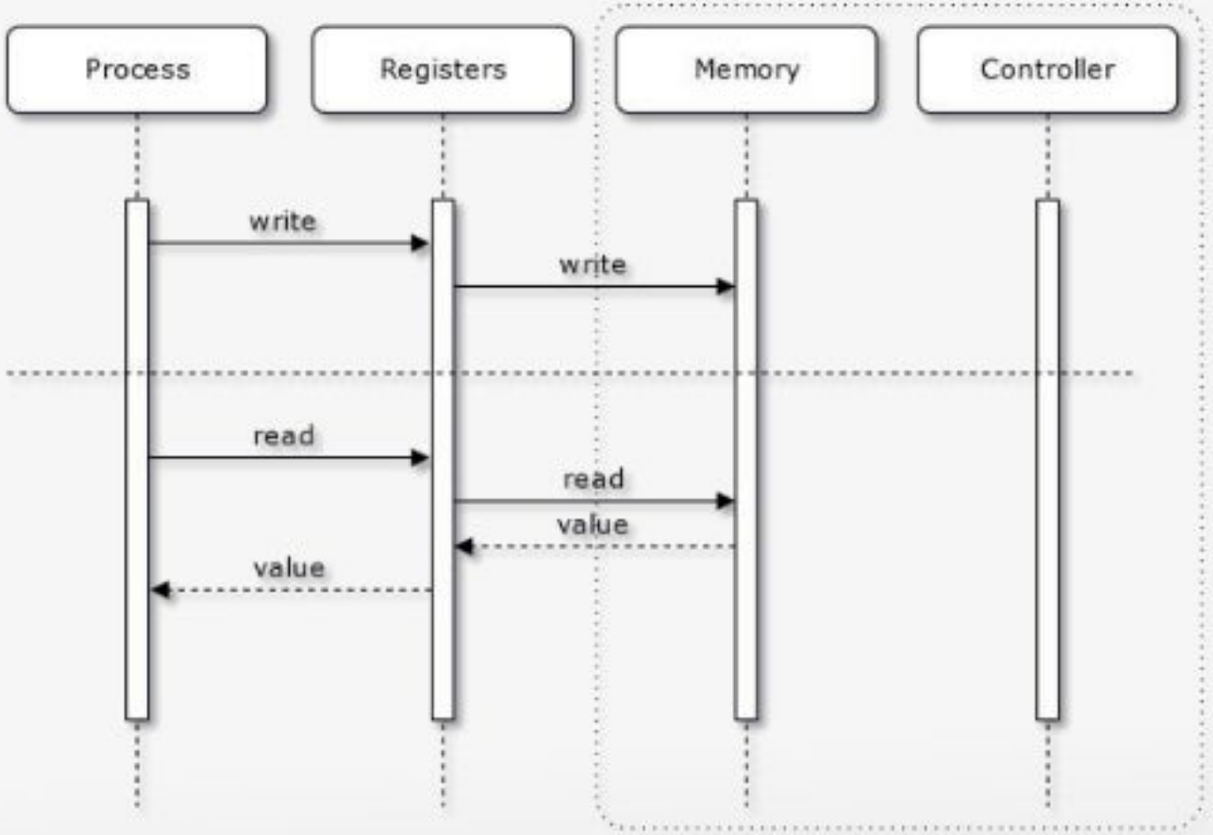
Пример: EFlags @ x86



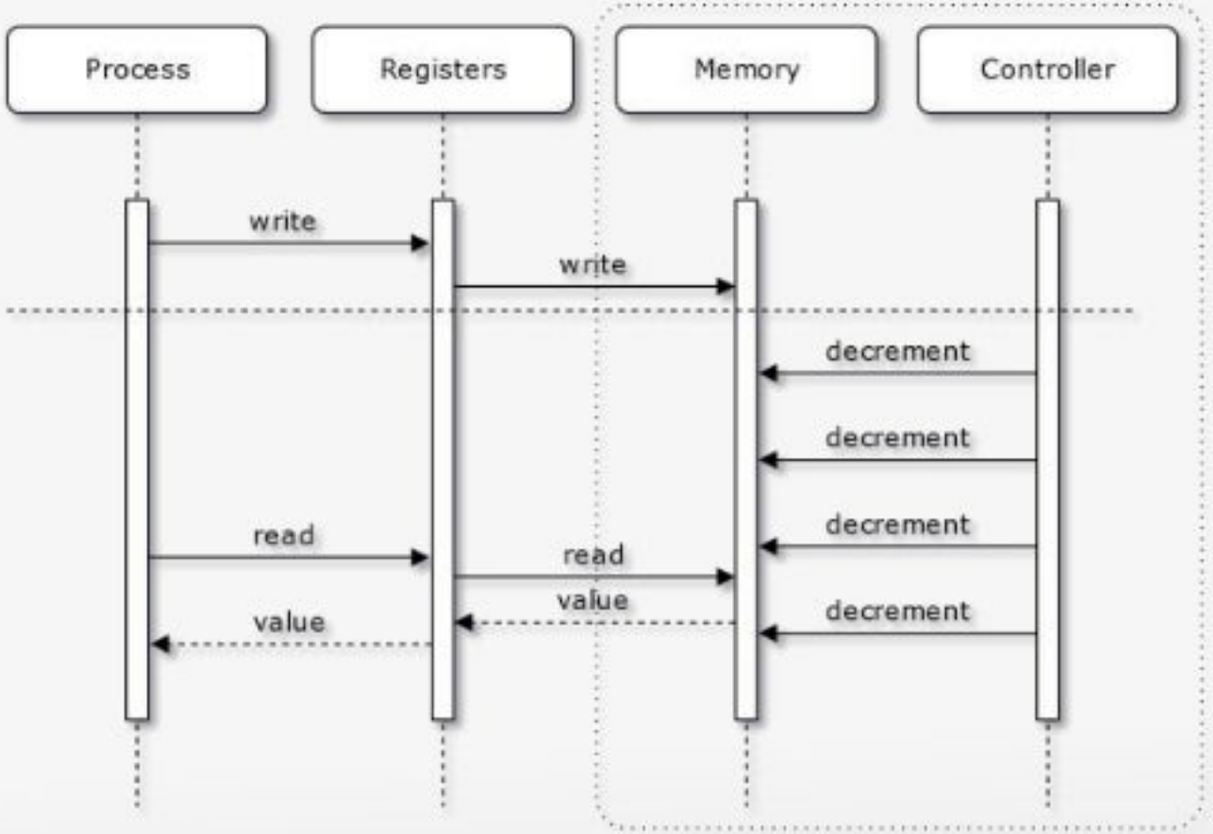
Модель ввода-вывода



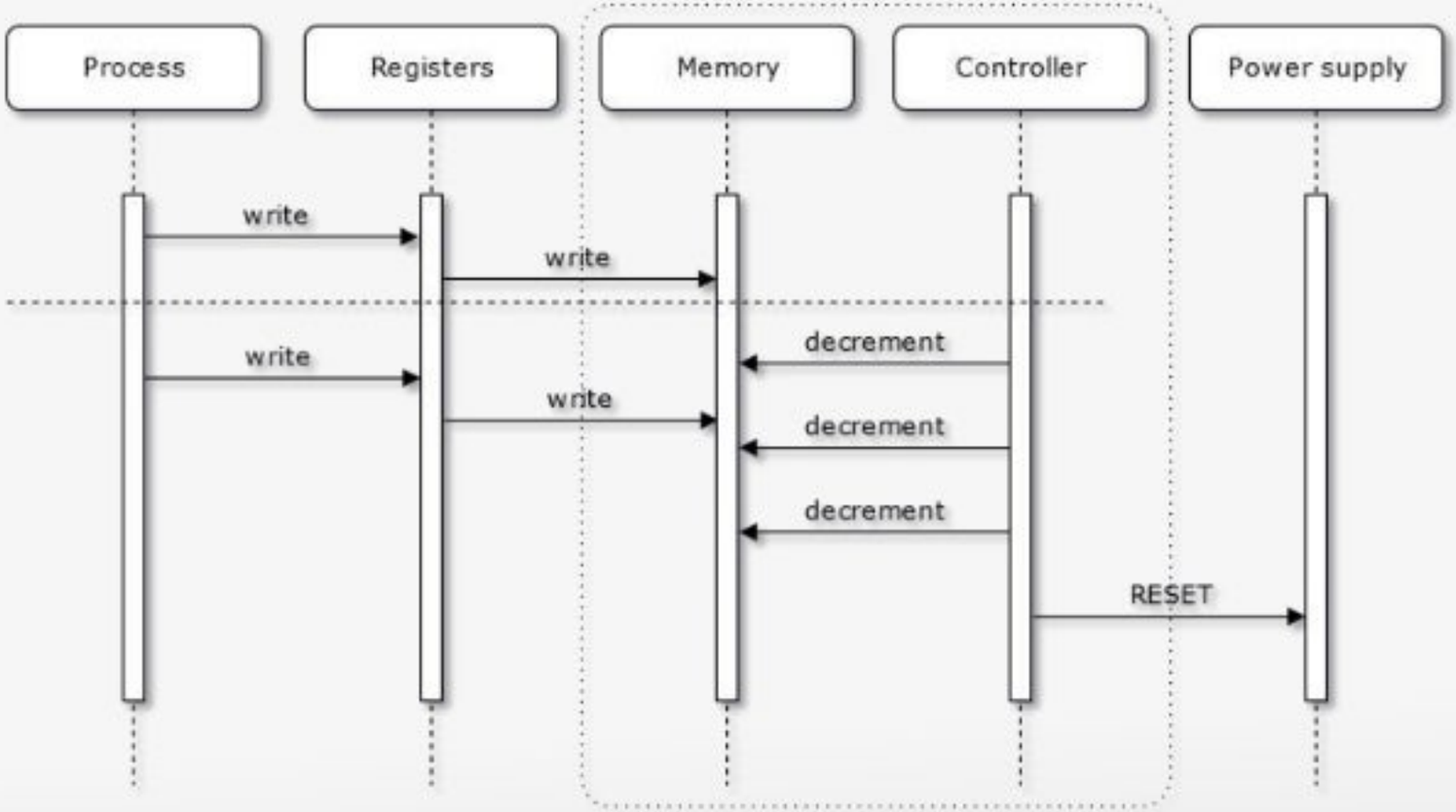
Регистры: ячейка памяти



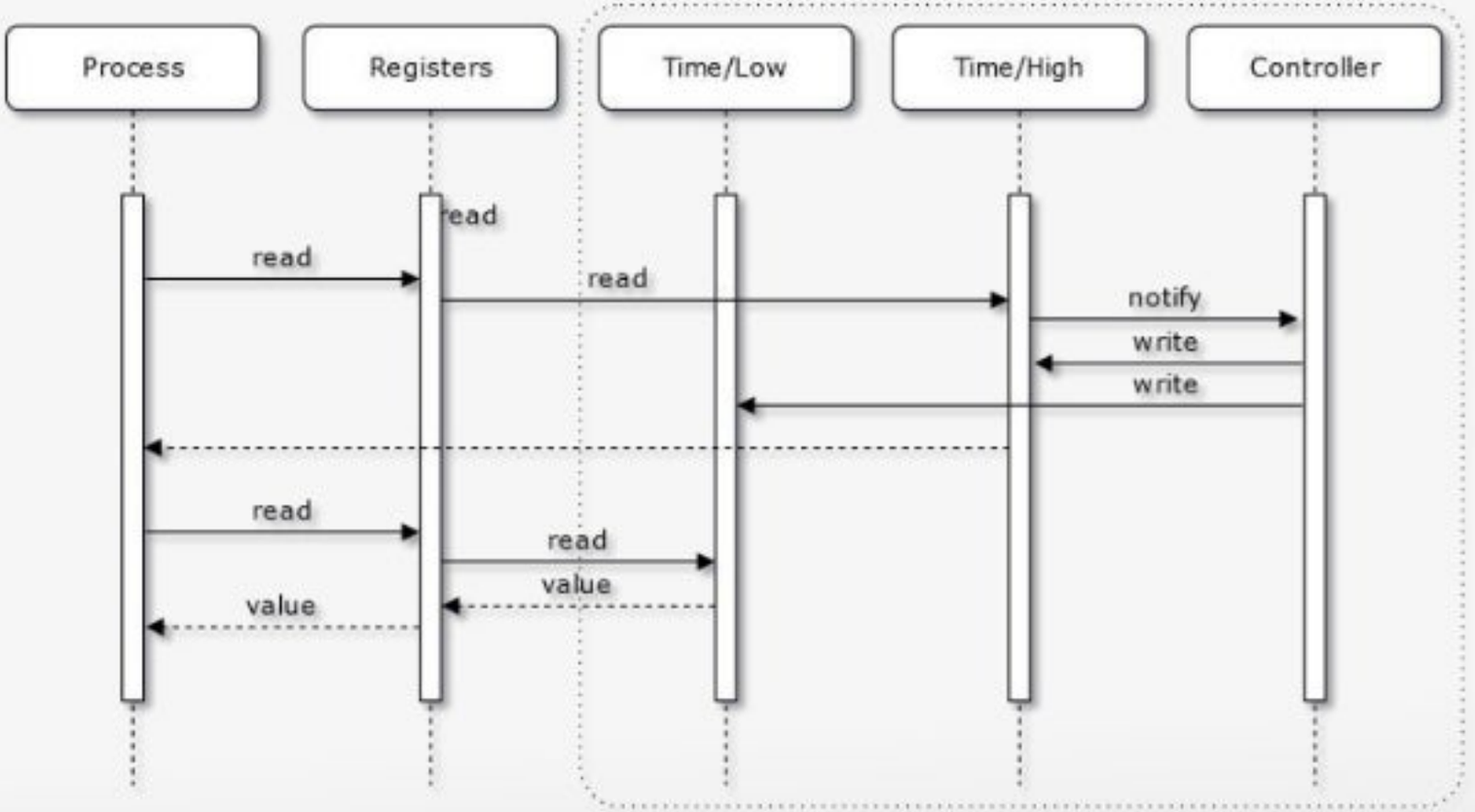
Регистры: счётчик



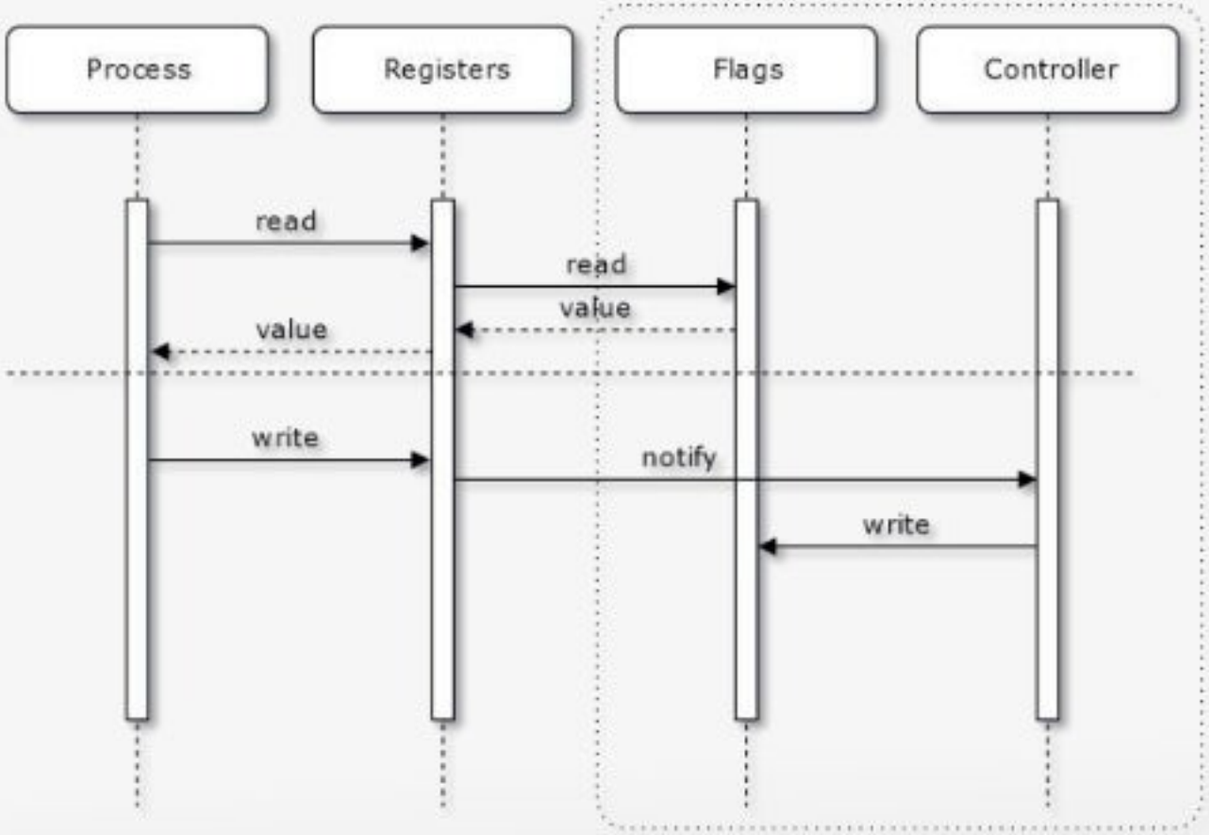
Регистры: сторожевой таймер

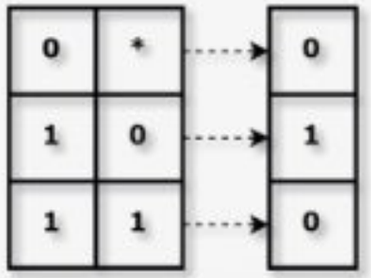


Регистры: защёлка



Регистры: статусные флаги





Исходные данные

- Подходы при взаимодействии периферийных устройств с программным обеспечением достаточно разнообразны
- Память, основанная на ячейках, хранящих информацию, не годится для моделирования всего многообразия аппаратных интерфейсов
- Язык реализации многих низкоуровневых драйверов для микроконтроллеров – Си, который имеет не слишком развитые средства абстрагирования, а если их реализовывать, поддержка абстрагирования часто является слишком дорогой из-за роста сложности и снижения быстродействия критического к производительности кода
- Многие драйверы уже написаны и в них не были заложены возможности тестирования.

Постановка задачи

- Моделирование поведения на уровне исходников. Полное моделирование поведения всего устройства не предполагается.
- Поддержка возможности встраивания в систему непрерывной интеграции для выполнения регрессионных тестов
- Работа под операционными системами Windows, Linux, macOS на архитектурах процессоров x86, x86_64, ARM (в т.ч. Thumb и Thumb2), ARM64, Aarch64 (при поддержке соответствующих ОС)
- Поддержка одновременной работы множества моделей контроллеров
- Простота реализации моделей контроллеров

Что имеем?

Операционные системы могут реагировать на обращение к диапазонам адресов виртуальной памяти:

- Подкачка
- Проецирование файлов в память
- Совместное использование библиотек
- Выделение памяти по требованию
- Механизм copy-on-write
- Ошибка доступа к памяти

Операционным системам для работы с виртуализацией памяти требуется наличие аппаратного MMU

Операционные системы транслируют аппаратные исключения доступа к памяти MMU (и не только их) в исключения или сигналы в контексте спровоцировавшего процесса

Существует возможность перехвата исключений/сигналов в пользовательском процессе

Что имеем?

	Windows	POSIX
Выделение памяти	VirtualAlloc	mmap
Настройка доступа	VirtualProtect	mprotect
Перехват	Structured Exception Handling	Signals
Контекст потока	Полный	Частичный

Перехваты чтения и записи

После записи регистра необходимо вызвать обработчик модели и выполнить обработку записанного пользовательским кодом в ячейку памяти значения

При чтении регистра необходимо вызвать обработчик в модели и предоставить пользовательскому коду значение запрашиваемой ячейки памяти

Порядок действий при перехвате

Перехват записи	Перехват чтения
При обращении к защищённой области памяти MMU формирует ошибку доступа, которую операционная система транслирует в сигнал SIGSEGV или исключение с кодом EXCEPTION_ACCESS_VIOLATION.	
	Снять защиту с регистрового файла
	Вызвать перехватчик чтения модели контроллера
Включить пошаговое выполнение или установить точку останова на следующей инструкции	
Снять защиту с регистрового файла	
Перезапустить инструкцию, вызвавшую исключение	
После завершения инструкции чтения/записи процессор сгенерирует событие трассировки, которое операционная система транслирует в сигнал SIGTRAP или исключение с кодом EXCEPTION_SINGLE_STEP	
Отключить трассировку/точку исключения	
Вызвать перехватчик записи модели контроллера	
Восстановить исходную защиту памяти и продолжить выполнение программы	

Другие аспекты

1. При начале работы контроллера может требоваться инициализация регистрового файла некоторыми начальными значениями
2. Генерация и обработка прерываний
3. Синхронизация доступа к регистровому файлу
4. Фоновая работа модели периферийного устройства может требовать особой работы с регистровым файлом
5. Необходима поддержка отладки как пользовательского кода (драйвера), так и модели контроллера

Требования к клиентскому коду

1. Регистровый файл должен быть объявлен как volatile-память
2. Пользовательский код не должен полагаться на абсолютные адреса областей памяти
3. Обработчик прерывания должен быть реализован в отдельной функции, которая может быть вызвана как из обработчика исключения (часто оформленного в виде макроса ISR), так и как функция обратного вызова из библиотечного кода

Библиотека ExMM

Библиотека реализует всё, что описано выше

Меньше 2000 строк кода!

Open-source, размещена на Github по адресу:

https://github.com/Alexey-Tkachenko/ExMM/

Скоро в виде Conan-пакета

Начало работы

1. Подключить библиотеку:

```
#include <exmm.hpp>
using namespace ExMM;
```

2. Определить структуру регистров:

```
struct Registers
{
    volatile int A;
    volatile int B;
    volatile int C;
};
```

Начало работы

3. Реализовать класс модели контроллера:

```
struct MyController final : public ControllerBase<HookTypes::ReadWrite, Registers>
{
    void HookRead(volatile Registers* data, size_t offset) override
    {}

    void HookWrite(volatile Registers* data, size_t offset) override
    {}
};
```

Начало работы

4. Создать объект контроллера:

```
MyController controller;
```

5. Получить указатель на регистровый файл:

```
volatile Registers *registers = controller.GetIoArea();
```

6. Запустить код в управляемом блоке:

```
ExMM::Run([&registers]()
{
    registers->A = 42;
    registers->B = 123;
    registers->C = -1;
});
```

Простой перехватчик

```
struct MyController final : public ControllerBase<HookTypes::ReadWrite, Registers>
{
    void HookRead(volatile Registers* data, size_t offset) override
    {
        std::cout << "Before read at offset " << std::hex << offset << std::endl;
    }

    void HookWrite(volatile Registers* data, size_t offset) override
    {
        std::cout << "After write at offset " << std::hex << offset << std::endl;
    }
};
```

Перехватчик, реализующий защёлку

```
struct Registers
{
    volatile uint32_t TimeLo; // Microseconds
    volatile uint32_t TimeHi; // Integral seconds
};

class LatchController final : public ControllerBase<HookTypes::Read, Registers>
{
    uint32_t latchedLoValue;
    std::chrono::high_resolution_clock::time_point whenStarted;
public:
    LatchController()
            : whenStarted(std::chrono::high_resolution_clock::now()),
              latchedLoValue()
    {}

    void HookRead(volatile Registers *data, size_t offset) override;
};
```

Перехватчик, реализующий защёлку

```
void LatchController::HookRead(volatile Registers *data, size_t offset)
{
    SwitchField(data, offset)
        .Case(&Registers::TimeLo,
              [this](auto &timeLo) { timeLo = latchedLoValue; })
        .Case(&Registers::TimeHi,
              [this](auto &timeHi) {
                  static const std::chrono::seconds oneSecond{1};
                  const auto elapsed = std::chrono::high_resolution_clock::now() - whenStarted;
                  const auto us = std::chrono::duration_cast<std::chrono::microseconds>(elapsed);
                  const auto seconds = us / oneSecond;

                  latchedLoValue = static_cast<uint32_t>((us % oneSecond).count());
                  timeHi = static_cast<uint32_t>(seconds);
              });
}
```

Поддержка массивов

```
struct Registers
{
    // ...
    volatile uint32_t Telemetry[4];
};

void SomeController::HookRead(volatile Registers* data, size_t offset)
{
    SwitchField(data, offset)
        .CaseArray(&Registers::Telemetry, [this](std::size_t index, auto& tm)
        {
            tm = shadowRecord.Words[index];
        });
}
```

Вложенные агрегаты

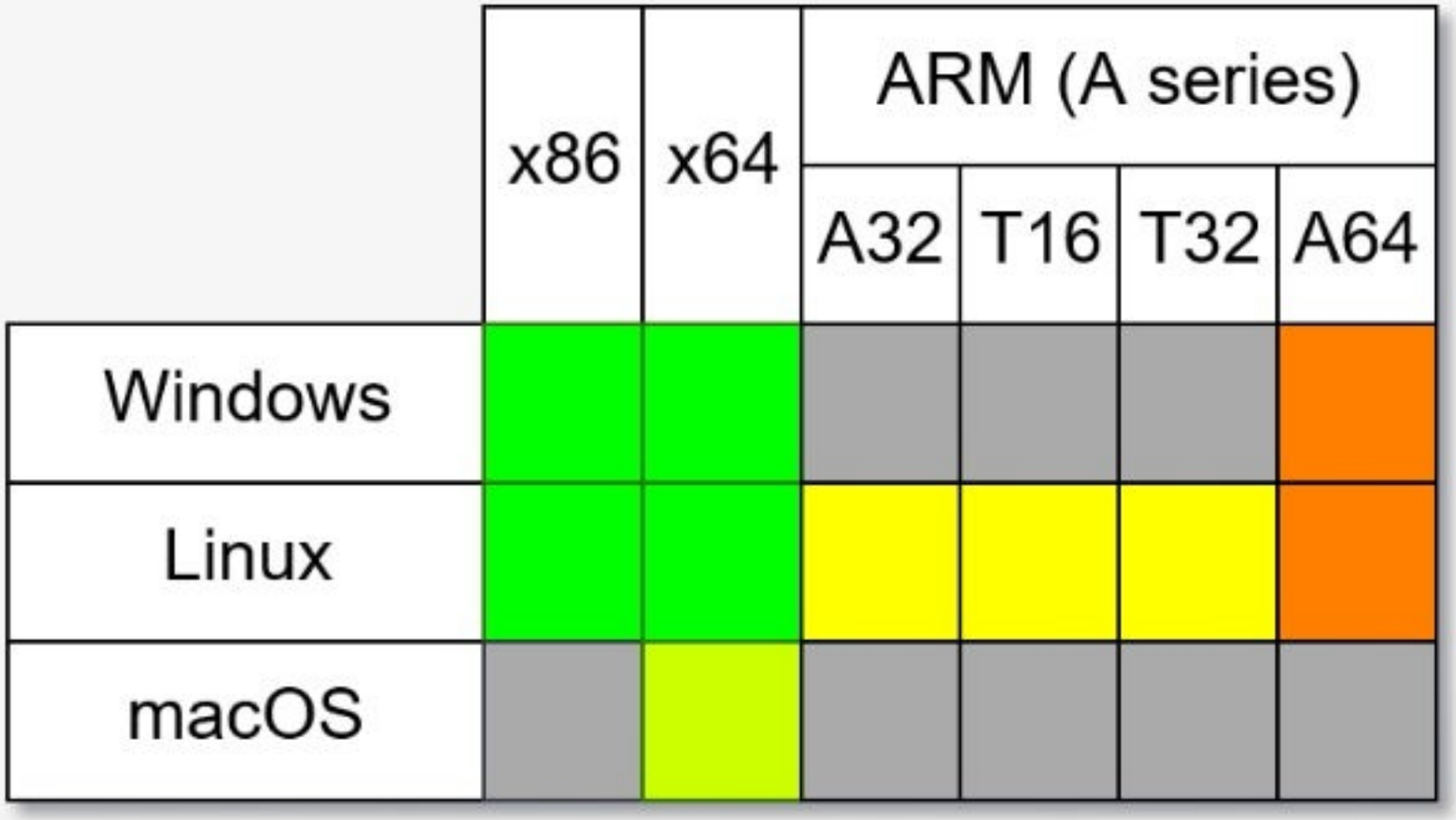
```
struct Registers
{
    struct TimerData
    {
        uint32_t TriggerTimeHi;         // Time when timer triggers, seconds.
        uint32_t TriggerTimeLo;         // Time when timer triggers, microseconds.
    };
    volatile TimerData Timers[8];       // Configuration.
};

void SomeController::HookWrite(volatile Registers *data, size_t offset)
{
    SwitchField(data, offset)
        .InsideArray(&Registers::Timers, [](std::size_t index, auto &next) {
            next.Case(&Registers::TimerData::TriggerTimeHi, [](auto &value) { ... })
                .Case(&Registers::TimerData::TriggerTimeLo, [](auto &value)
                    { value = std::min(uint32_t{value}, 999999u); }
                );
            });
}
```

Необработанный доступ

```
void SomeController::HookWrite(Registers *data, size_t offset)
{
    SwitchField(data, offset)
        .Case(&Registers::ControlWord, [this](auto &value) { ...})
        .Case(&Registers::StatusWord, [this](auto &value) { ... })
        .InsideArray(&Registers::Timers, [](std::size_t index, auto &next) { ... })
        .Else([data](size_t offset) {
            std::cout << "((not observed)) "
            << reinterpret_cast<volatile uint32_t *>(data)[offset / 4]
            << std::endl;
        });
}
```

Поддержка платформ



Известные проблемы

Необработанный доступ на Linux за пределами памяти контроллера завершает процесс

Низкая производительность DSL

Неполная документация

Дальнейшие планы

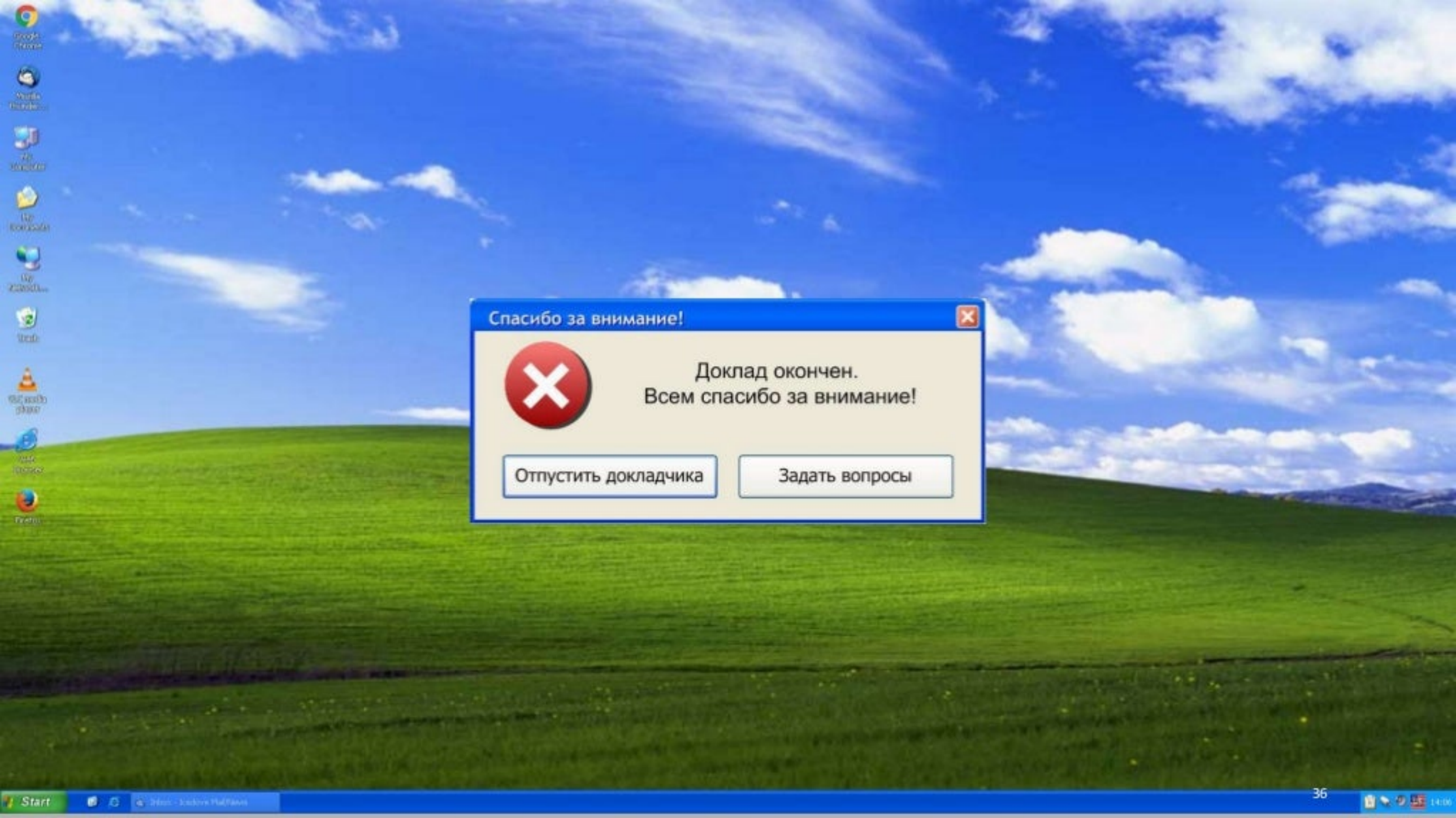
Поддержка оставшихся платформ

Развитие DSL обработки доступа

Библиотека примитивов

Средства управления спецификациями

Дописать документацию


Спасибо за внимание!
Доклад окончен.
Всем спасибо за внимание!
Отпустить докладчика
Задать вопросы
Start

Q&A

АЛЕКСЕЙ ТКАЧЕНКО

ОАО «ПЕЛЕНГ»

TKACHENKO@PELENG.BY, ALEXEY.TKACHENKO@GMAIL.COM

HTTPS://GITHUB.COM/ALEXEY-TKACHENKO/