DAP SDK Test Framework - Полное Руководство

Асинхронное тестирование, моки и автоматизация тестов

Команда разработки Cellframe

28 октября 2025

Содержание

1 Информация о документе **3**
1.1 История изменений . . . 3
1.2 Авторские права . . . 3
1.3 Лицензия . . . 3

2 Часть I: Введение **4**
2.1 1. Обзор . . . 4
2.1.1 1.1 Что такое DAP SDK Test Framework? . . . 4
2.1.2 1.2 Зачем использовать этот фреймворк? . . . 4
2.1.3 1.3 Ключевые возможности . . . 4
2.1.4 1.4 Быстрое сравнение . . . 5
2.1.5 1.5 Целевая аудитория . . . 5
2.1.6 1.6 Предварительные требования . . . 5
2.2 2. Быстрый Старт . . . 6
2.2.1 2.1 Первый тест (5 минут) . . . 6
2.2.2 2.2 Добавление async таймаута (2 минуты) . . . 6
2.2.3 2.3 Добавление моков (5 минут) . . . 7
2.3 3. Справочник API . . . 9
2.3.1 3.1 Async Testing API . . . 9
2.3.2 3.2 Mock Framework API . . . 10
2.3.3 3.3 API пользовательских линкер-оберток . . . 13
2.3.4 3.4 Интеграция с CMake . . . 14
2.3.5 3.5 Асинхронное выполнение моков . . . 16
2.4 4. Полные примеры . . . 19
2.4.1 4.1 Тест стейт-машины (Пример из реального проекта) . . . 19
2.4.2 4.2 Мок с callback . . . 20
2.4.3 4.3 Мок с задержками выполнения . . . 20
2.4.4 4.4 Пользовательская линкер-обертка (Продвинутый уровень) 21
2.4.5 4.5 Динамическое поведение мока . . . 22
2.4.6 4.6 Мокирование в статических библиотеках . . . 23
2.4.7 4.7 Асинхронное выполнение моков . . . 25
2.5 5. Глоссарий . . . 28
2.6 6. Решение проблем . . . 29
2.6.1 Проблема: Тест зависает бесконечно . . . 29
2.6.2 Проблема: Высокая загрузка CPU . . . 29
2.6.3 Проблема: Мок не вызывается (выполняется реальная функция) . . . 29

2.6.4 Проблема: Неправильное возвращаемое значение 29
2.6.5 Проблема: Нестабильные тесты (периодические сбои) 29
2.6.6 Проблема: Ошибка компиляции “undefined reference to __wrap” 30
2.6.7 Проблема: Callback мока не выполняется 30
2.6.8 Проблема: Мок не работает для функций в статической библиотеке 30
2.6.9 Проблема: Ошибка линкера “multiple definition” 31
2.6.10 Проблема: Задержка не работает 31

1 Информация о документе

Версия: 1.0.1
Дата: 28 октября 2025
Статус: Production Ready
Язык: Русский

1.1 История изменений

Версия	Дата	Изменения	Автор
1.0.1	2025-10-28	Обновлены примеры, улучшен справочник API, добавлено решение проблем	Команда Cellframe
1.0.0	2025-10-27	Первая версия полного руководства	Команда Cellframe

1.2 Авторские права

Этот документ описывает DAP SDK Test Framework, часть проекта Cellframe Network.

1.3 Лицензия

См. файл LICENSE проекта для условий использования.

2 Часть I: Введение

2.1 1. Обзор

DAP SDK Test Framework - это production-ready инфраструктура тестирования для экосистемы блокчейна Cellframe. Она предоставляет комплексные инструменты для тестирования асинхронных операций, мокирования внешних зависимостей и обеспечения надёжного выполнения тестов на разных платформах.

2.1.1 1.1 Что такое DAP SDK Test Framework?

Полное решение для тестирования, включающее:

- **Async Testing Framework** - Инструменты для тестирования асинхронных операций с таймаутами
- **Mock Framework** - Мокирование функций без модификации кода
- **Async Mock Execution** - Асинхронное выполнение моков с пулом потоков
- **Auto-Wrapper System** - Автоматическая конфигурация линкера
- **Self-Tests** - 21 тест-функция, валидирующая надёжность фреймворка

2.1.2 1.2 Зачем использовать этот фреймворк?

Проблема: Тестирование асинхронного кода сложно - Операции завершаются в непредсказуемое время - Сетевые задержки варьируются - Тесты могут зависать бесконечно - Внешние зависимости усложняют тестирование

Решение: Этот фреймворк предоставляет - [x] Защиту от зависаний (глобальный + для каждой операции) - [x] Эффективное ожидание (polling + condition variables) - [x] Изоляцию зависимостей (мокирование) - [x] Реалистичную симуляцию (задержки, ошибки) - [x] Потокобезопасные операции - [x] Кроссплатформенность

2.1.3 1.3 Ключевые возможности

Возможность	Описание	Польза
Global Timeout	alarm + siglongjmp	Предотвращает зависание CI/CD
Condition Polling	Конфигурируемые интервалы	Эффективное ожидание
pthread Helpers	Обёртки для condition variables	Потокобезопасная координация
Mock Framework	На основе линкера (`--wrap`)	Нулевой техдолг
Async Mocks	Выполнение в thread pool	Реальная симуляция async поведения
Задержки	Fixed, Range, Variance	Реалистичная симуляция времени
Callbacks	Inline + Runtime	Динамическое поведение моков

Возможность	Описание	Польза
Auto-Wrapper	Bash/PowerShell скрипты	Автоматическая настройка
Self-Tests	21 тест-функция	Проверенная надёжность

2.1.4 1.4 Быстрое сравнение

Традиционный подход:

```
// [!] Плохо: занятое ожидание, нет таймаута, трата CPU
while (!done) {
    usleep(10000);  // 10ms сон
}
```

C DAP Test Framework:

```
// [+] Хорошо: эффективно, защита таймаутом, автоматическое логирование
DAP_TEST_WAIT_UNTIL(done == true, 5000, "Should complete");
```

2.1.5 1.5 Целевая аудитория

- Разработчики DAP SDK
- Контрибьюторы Cellframe SDK
- Разработчики VPN Client
- Все, кто тестирует асинхронный C код в экосистеме Cellframe

2.1.6 1.6 Предварительные требования

Необходимые знания: - Программирование на C - Базовое понимание асинхронных операций - Основы CMake - Концепции pthread (для продвинутых возможностей)

Необходимое ПО: - GCC 7+ или Clang 10+ (или MinGW на Windows) - CMake 3.10+ - Библиотека pthread - Linux, macOS, или Windows (частичная поддержка)

2.2 2. Быстрый Старт

2.2.1 2.1 Первый тест (5 минут)

Шаг 1: Создайте файл теста

```
// my_test.c
#include "dap_test.h"
#include "dap_common.h"

#define LOG_TAG "my_test"

int main() {
    dap_common_init("my_test", NULL);

    // Код теста
    int result = 2 + 2;
    dap_assert_PIF(result == 4, "Math should work");

    log_it(L_INFO, "[+] Тест пройден!");

    dap_common_deinit();
    return 0;
}
```

Шаг 2: Создайте CMakeLists.txt

```
add_executable(my_test my_test.c)
target_link_libraries(my_test dap_core)
add_test(NAME my_test COMMAND my_test)
```

Шаг 3: Соберите и запустите

```
cd build
cmake ..
make my_test
./my_test
```

2.2.2 2.2 Добавление async таймаута (2 минуты)

```
#include "dap_test.h"
#include "dap_test_async.h"
#include "dap_common.h"

#define LOG_TAG "my_test"
#define TIMEOUT_SEC 30

int main() {
    dap_common_init("my_test", NULL);

    // Добавьте глобальный таймаут
    dap_test_global_timeout_t timeout;
    if (dap_test_set_global_timeout(&timeout, TIMEOUT_SEC, "My Test")) {
        return 1;  // Таймаут сработал
    }
```

```
    // Ваши тесты здесь

    dap_test_cancel_global_timeout();
    dap_common_deinit();
    return 0;
}
```

Обновите CMakeLists.txt:

```
# Подключите библиотеку test-framework (включает dap_test, dap_mock и т.д.)
target_link_libraries(my_test dap_test dap_core pthread)
```

2.2.3 2.3 Добавление моков (5 минут)

```
#include "dap_test.h"
#include "dap_mock.h"
#include "dap_common.h"
#include <assert.h>

#define LOG_TAG "my_test"

// Объявите мок
DAP_MOCK_DECLARE(external_api_call);

int main() {
    dap_common_init("my_test", NULL);
    // Примечание: dap_mock_init() не нужен - авто-инициализация!

    // Настройте мок на возврат 42
    DAP_MOCK_SET_RETURN(external_api_call, (void*)42);

    // Запустите код, который вызывает external_api_call
    int result = my_code_under_test();

    // Проверьте что мок был вызван один раз и вернул правильное значение
    assert(DAP_MOCK_GET_CALL_COUNT(external_api_call) == 1);
    assert(result == 42);

    log_it(L_INFO, "[+] Тест пройден!");

    // Опциональная очистка (если нужно сбросить моки)
    // dap_mock_deinit();
    dap_common_deinit();
    return 0;
}
```

Обновите CMakeLists.txt:

```
include(${CMAKE_CURRENT_SOURCE_DIR}/../test-framework/mocks/DAPMockAutoWrap.cmak

# Подключите библиотеку test-framework (включает dap_test, dap_mock и т.д.)
target_link_libraries(my_test dap_test dap_core pthread)
```

```
# Автогенерация --wrap флагов линкера
dap_mock_autowrap(my_test)

# Если нужно мокировать функции в статических библиотеках:
# dap_mock_autowrap_with_static(my_test dap_static_lib)
```

2.3 3. Справочник API

2.3.1 3.1 Async Testing API

2.3.1.1 Глобальный таймаут

```
int dap_test_set_global_timeout(
    dap_test_global_timeout_t *a_timeout,
    uint32_t a_timeout_sec,
    const char *a_test_name
);
// Возвращает: 0 при настройке, 1 если таймаут сработал

void dap_test_cancel_global_timeout(void);
```

2.3.1.2 Опрос условий

```
bool dap_test_wait_condition(
    dap_test_condition_cb_t a_condition,
    void *a_user_data,
    const dap_test_async_config_t *a_config
);
// Возвращает: true если условие выполнено, false при таймауте
//
// Сигнатура callback:
// typedef bool (*dap_test_condition_cb_t)(void *a_user_data);
//
// Структура конфигурации:
// typedef struct {
//     uint32_t timeout_ms;          // Макс. время ожидания (мс)
//     uint32_t poll_interval_ms;    // Интервал опроса (мс)
//     bool fail_on_timeout;         // abort() при таймауте?
//     const char *operation_name;   // Для логирования
// } dap_test_async_config_t;
//
// Дефолтная конфигурация: DAP_TEST_ASYNC_CONFIG_DEFAULT
//   - timeout_ms: 5000 (5 секунд)
//   - poll_interval_ms: 100 (100 мс)
//   - fail_on_timeout: true
//   - operation_name: "async operation"
```

2.3.1.3 pthread хелперы

```
void dap_test_cond_wait_init(dap_test_cond_wait_ctx_t *a_ctx);
bool dap_test_cond_wait(dap_test_cond_wait_ctx_t *a_ctx, uint32_t a_timeout_ms);
void dap_test_cond_signal(dap_test_cond_wait_ctx_t *a_ctx);
void dap_test_cond_wait_deinit(dap_test_cond_wait_ctx_t *a_ctx);
```

2.3.1.4 Утилиты времени

```
uint64_t dap_test_get_time_ms(void);  // Монотонное время в мс
void dap_test_sleep_ms(uint32_t a_delay_ms);  // Кроссплатформенный sleep
```

2.3.1.5 Макросы

```
DAP_TEST_WAIT_UNTIL(condition, timeout_ms, msg)
// Быстрое ожидание условия
```

2.3.2 3.2 Mock Framework API

Заголовочный файл: dap_mock.h

2.3.2.1 Инициализация фреймворка

```
int dap_mock_init(void);
// Опционально: переинициализация мок-фреймворка (авто-инициализируется через ко
// Возвращает: 0 при успехе
// Примечание: Фреймворк авто-инициализируется до main(), ручной вызов не требуе
// Кроссплатформенность: использует __attribute__((constructor)) на GCC/Clang/Mi
//                       статический C++ объект на MSVC

void dap_mock_deinit(void);
// Очистка мок-фреймворка (вызывать в teardown при необходимости)
// Примечание: Также авто-деинициализирует async систему если она была включена
// Авто-очистка: использует __attribute__((destructor)) на GCC/Clang,
//               atexit() на MSVC для автоматической очистки после main()
```

2.3.2.2 Макросы объявления моков Простое объявление (авто-включено, возврат 0):

```
DAP_MOCK_DECLARE(function_name);
```

С конфигурационной структурой:

```
DAP_MOCK_DECLARE(function_name, {
    .enabled = true,
    .return_value.l = 0xDEADBEEF,
    .delay = {
        .type = DAP_MOCK_DELAY_FIXED,
        .fixed_us = 1000
    }
});
```

Со встроенным callback:

```
DAP_MOCK_DECLARE(function_name, {.return_value.i = 0}, {
    // Тело callback - пользовательская логика для каждого вызова
    if (a_arg_count >= 1) {
        int arg = (int)(intptr_t)a_args[0];
        return (void*)(intptr_t)(arg * 2);  // Удваиваем входное значение
    }
    return (void*)0;
});
```

Для пользовательской обертки (без авто-генерации):

```
DAP_MOCK_DECLARE_CUSTOM(function_name, {
    .delay = {
```

```
        .type = DAP_MOCK_DELAY_VARIANCE,
        .variance = {.center_us = 100000, .variance_us = 50000}
    }
});
```

2.3.2.3 Конфигурационные структуры dap_mock_config_t:

```
typedef struct dap_mock_config {
    bool enabled;                          // Включить/выключить мок
    dap_mock_return_value_t return_value;  // Возвращаемое значение
    dap_mock_delay_t delay;                // Задержка выполнения
    bool async;                            // Выполнять callback асинхронно (default
    bool call_original_before;            // Вызвать оригинальную функцию ДО мок-лог
    bool call_original_after;             // Вызвать оригинальную функцию ПОСЛЕ мок-
} dap_mock_config_t;

// По умолчанию: enabled=true, return=0, без задержки, sync, без вызова оригинал
#define DAP_MOCK_CONFIG_DEFAULT { \
    .enabled = true, \
    .return_value = {0}, \
    .delay = {.type = DAP_MOCK_DELAY_NONE}, \
    .async = false, \
    .call_original_before = false, \
    .call_original_after = false \
}

// Passthrough конфигурация: отслеживание вызовов, но всегда вызывается оригинал
#define DAP_MOCK_CONFIG_PASSTHROUGH { \
    .enabled = true, \
    .return_value = {0}, \
    .delay = {.type = DAP_MOCK_DELAY_NONE}, \
    .async = false, \
    .call_original_before = true, \
    .call_original_after = false \
}
```

dap_mock_return_value_t:

```
typedef union dap_mock_return_value {
    int i;          // Для int, bool, малых типов
    long l;         // Для указателей (приведение через intptr_t)
    uint64_t u64;   // Для uint64_t, size_t (64-бит)
    void *ptr;      // Для void*, общих указателей
    char *str;      // Для char*, строк
} dap_mock_return_value_t;
```

dap_mock_delay_t:

```
typedef enum {
    DAP_MOCK_DELAY_NONE,       // Без задержки
    DAP_MOCK_DELAY_FIXED,      // Фиксированная задержка
    DAP_MOCK_DELAY_RANGE,      // Случайная в [min, max]
    DAP_MOCK_DELAY_VARIANCE    // Центр ± разброс
} dap_mock_delay_type_t;
```

```
typedef struct dap_mock_delay {
    dap_mock_delay_type_t type;
    union {
        uint64_t fixed_us;
        struct { uint64_t min_us; uint64_t max_us; } range;
        struct { uint64_t center_us; uint64_t variance_us; } variance;
    };
} dap_mock_delay_t;
```

2.3.2.4 Макросы управления

```
DAP_MOCK_ENABLE(func_name)
// Включить мок (перехват вызовов)
// Пример: DAP_MOCK_ENABLE(dap_stream_write);

DAP_MOCK_DISABLE(func_name)
// Выключить мок (вызов реальной функции)
// Пример: DAP_MOCK_DISABLE(dap_stream_write);

DAP_MOCK_RESET(func_name)
// Сбросить историю вызовов и статистику
// Пример: DAP_MOCK_RESET(dap_stream_write);

DAP_MOCK_SET_RETURN(func_name, value)
// Установить возвращаемое значение (приведение через (void*) или (void*)(intptr
// Пример: DAP_MOCK_SET_RETURN(dap_stream_write, (void*)(intptr_t)42);

DAP_MOCK_GET_CALL_COUNT(func_name)
// Получить количество вызовов мока (возвращает int)
// Пример: int count = DAP_MOCK_GET_CALL_COUNT(dap_stream_write);

DAP_MOCK_WAS_CALLED(func_name)
// Возвращает true если был вызван хотя бы раз (возвращает bool)
// Пример: assert(DAP_MOCK_WAS_CALLED(dap_stream_write));

DAP_MOCK_GET_ARG(func_name, call_idx, arg_idx)
// Получить конкретный аргумент из конкретного вызова
// call_idx: 0-базированный индекс вызова (0 = первый вызов)
// arg_idx: 0-базированный индекс аргумента (0 = первый аргумент)
// Возвращает: void* (приведите к нужному типу)
// Пример: void *buffer = DAP_MOCK_GET_ARG(dap_stream_write, 0, 1);
//          size_t size = (size_t)DAP_MOCK_GET_ARG(dap_stream_write, 0, 2);
```

2.3.2.5 Макросы конфигурации задержек

```
DAP_MOCK_SET_DELAY_FIXED(func_name, microseconds)
DAP_MOCK_SET_DELAY_MS(func_name, milliseconds)
// Установить фиксированную задержку

DAP_MOCK_SET_DELAY_RANGE(func_name, min_us, max_us)
DAP_MOCK_SET_DELAY_RANGE_MS(func_name, min_ms, max_ms)
```

```
// Установить случайную задержку в диапазоне

DAP_MOCK_SET_DELAY_VARIANCE(func_name, center_us, variance_us)
DAP_MOCK_SET_DELAY_VARIANCE_MS(func_name, center_ms, variance_ms)
// Установить задержку с разбросом (например, 100мс ± 20мс)

DAP_MOCK_CLEAR_DELAY(func_name)
// Убрать задержку
```

2.3.2.6 Конфигурация callback

```
DAP_MOCK_SET_CALLBACK(func_name, callback_func, user_data)
// Установить пользовательскую функцию callback

DAP_MOCK_CLEAR_CALLBACK(func_name)
// Убрать callback (использовать return_value)

// Сигнатура callback:
typedef void* (*dap_mock_callback_t)(
    void **a_args,
    int a_arg_count,
    void *a_user_data
);
```

2.3.3 3.3 API пользовательских линкер-оберток

Заголовочный файл: dap_mock_linker_wrapper.h

2.3.3.1 Макрос DAP_MOCK_WRAPPER_CUSTOM Создает пользовательскую линкер-обертку с PARAM синтаксисом:

```
DAP_MOCK_WRAPPER_CUSTOM(return_type, function_name,
    PARAM(type1, name1),
    PARAM(type2, name2),
    ...
) {
    // Реализация пользовательской обертки
}
```

Возможности: - Автоматически генерирует сигнатуру функции - Автоматически создает массив void* аргументов с правильным приведением типов - Автоматически проверяет, включен ли мок - Автоматически выполняет настроенную задержку - Автоматически записывает вызов - Вызывает реальную функцию при выключенном моке

Пример:

```
DAP_MOCK_WRAPPER_CUSTOM(int, my_function,
    PARAM(const char*, path),
    PARAM(int, flags),
    PARAM(mode_t, mode)
) {
    // Ваша пользовательская логика здесь
```

```
    if (strcmp(path, "/dev/null") == 0) {
        return -1;  // Симуляция ошибки
    }
    return 0;  // Успех
}
```

Макрос PARAM: - Формат: `PARAM(type, name)` - Автоматически извлекает тип и имя - Правильно обрабатывает приведение к void* - Использует `uintptr_t` для безопасного приведения указателей и целочисленных типов

2.3.3.2 Упрощенные макросы оберток

Для распространенных типов возвращаемых значений:

```
DAP_MOCK_WRAPPER_INT(func_name, (params), (args))
DAP_MOCK_WRAPPER_PTR(func_name, (params), (args))
DAP_MOCK_WRAPPER_VOID_FUNC(func_name, (params), (args))
DAP_MOCK_WRAPPER_BOOL(func_name, (params), (args))
DAP_MOCK_WRAPPER_SIZE_T(func_name, (params), (args))
```

2.3.4 3.4 Интеграция с CMake

CMake модуль: `mocks/DAPMockAutoWrap.cmake`

```
include(${CMAKE_SOURCE_DIR}/dap-sdk/test-framework/mocks/DAPMockAutoWrap.cmake)

# Автоматическое сканирование исходников и генерация --wrap флагов
dap_mock_autowrap(target_name)

# Альтернатива: явно указать исходные файлы
dap_mock_autowrap(TARGET target_name SOURCE file1.c file2.c)
```

Как работает: 1. Сканирует исходные файлы на наличие паттернов `DAP_MOCK_DECLARE` 2. Извлекает имена функций 3. Добавляет `-Wl,--wrap=function_name` к флагам линкера 4. Работает с GCC, Clang, MinGW

2.3.4.1 Мокирование функций в статических библиотеках

Проблема: При линковке статических библиотек (`lib*.a`) функции могут быть исключены из финального исполняемого файла, если они не используются напрямую. Это приводит к тому, что `--wrap` флаги не работают для функций внутри статических библиотек.

Решение: Используйте функцию `dap_mock_autowrap_with_static()` для оборачивания статических библиотек флагами `--whole-archive`, что заставляет линкер включить все символы из статической библиотеки.

Пример использования:

```
include(${CMAKE_SOURCE_DIR}/dap-sdk/test-framework/mocks/DAPMockAutoWrap.cmake)

add_executable(test_http_client
    test_http_client.c
    test_http_client_mocks.c
)
```

```
# Обычная линковка
target_link_libraries(test_http_client
    dap_test           # Test framework
    dap_core           # Core library
    dap_http_server    # Статическая библиотека, которую нужно мокировать
    pthread
)

# Автогенерация --wrap флагов из исходников теста
dap_mock_autowrap(test_http_client)

# Важно: обернуть статическую библиотеку --whole-archive ПОСЛЕ dap_mock_autowrap
# Это заставляет линкер включить все символы из dap_http_server,
# включая те, которые используются только внутри библиотеки
dap_mock_autowrap_with_static(test_http_client dap_http_server)
```

Что делает dap_mock_autowrap_with_static: 1. Перестраивает список линкуемых библиотек 2. Оборачивает указанные статические библиотеки флагами: - `-Wl,--whole-archive` (перед библиотекой) - <library_name> (сама библиотека) - `-Wl,--no-whole-archive` (после библиотеки) 3. Добавляет `-Wl,--allow-multiple-definition` для обработки дублирующихся символов

Важные замечания:

1. **Порядок вызовов важен:**

   ```
   # Правильно:
   dap_mock_autowrap(test_target)                    # Сначала автогенерация
   dap_mock_autowrap_with_static(test_target lib)    # Потом --whole-archive

   # Неправильно:
   dap_mock_autowrap_with_static(test_target lib)    # Это перезапишет предыду
   dap_mock_autowrap(test_target)
   ```

2. **Множественные библиотеки:**

   ```
   # Можно обернуть несколько статических библиотек сразу
   dap_mock_autowrap_with_static(test_target
       dap_http_server
       dap_stream
       dap_crypto
   )
   ```

3. **Ограничения:**

   - Работает только с GCC, Clang и MinGW
   - Может увеличить размер исполняемого файла
   - Не используйте для shared библиотек (`.so`, `.dll`)

Пример полной конфигурации:

```
include(${CMAKE_SOURCE_DIR}/dap-sdk/test-framework/mocks/DAPMockAutoWrap.cmake)

add_executable(test_stream_mocks
    test_stream_mocks.c
    test_stream_mocks_wrappers.c
```

```
)

target_link_libraries(test_stream_mocks
    dap_test
    dap_stream       # Статическая библиотека
    dap_net          # Статическая библиотека
    dap_core
    pthread
)

target_include_directories(test_stream_mocks PRIVATE
    ${CMAKE_SOURCE_DIR}/dap-sdk/test-framework
    ${CMAKE_SOURCE_DIR}/dap-sdk/core/include
)

# Автогенерация --wrap флагов
dap_mock_autowrap(test_stream_mocks)

# Оборачивание статических библиотек для мокирования внутренних функций
dap_mock_autowrap_with_static(test_stream_mocks
    dap_stream
    dap_net
)
```

Проверка правильности настройки:

```
# Проверьте флаги линкера
cd build
make VERBOSE=1 | grep -E "--wrap|--whole-archive"

# Должно быть:
# -Wl,--wrap=dap_stream_write
# -Wl,--wrap=dap_net_tun_create
# -Wl,--whole-archive ... dap_stream ... -Wl,--no-whole-archive
# -Wl,--whole-archive ... dap_net ... -Wl,--no-whole-archive
```

2.3.5 3.5 Асинхронное выполнение моков

Заголовок: `dap_mock_async.h`

Предоставляет легковесное асинхронное выполнение mock callback'ов без необходимости полной инфраструктуры dap_events. Идеально для unit тестов, требующих симуляции async поведения в изоляции.

2.3.5.1 Инициализация

```
// Инициализация async системы с worker потоками
int dap_mock_async_init(uint32_t a_worker_count);
// a_worker_count: 0 = auto, обычно 1-2 для unit тестов
// Возвращает: 0 при успехе

// Деинициализация (ждёт завершения всех задач)
void dap_mock_async_deinit(void);
```

```c
// Проверка инициализации
bool dap_mock_async_is_initialized(void);
```

2.3.5.2 Планирование задач

```c
// Запланировать выполнение async callback
dap_mock_async_task_t* dap_mock_async_schedule(
    dap_mock_async_callback_t a_callback,
    void *a_arg,
    uint32_t a_delay_ms  // 0 = немедленно
);

// Отменить pending задачу
bool dap_mock_async_cancel(dap_mock_async_task_t *a_task);
```

2.3.5.3 Ожидание завершения

```c
// Ждать конкретную задачу
bool dap_mock_async_wait_task(
    dap_mock_async_task_t *a_task,
    int a_timeout_ms  // -1 = бесконечно, 0 = не ждать
);

// Ждать все pending задачи
bool dap_mock_async_wait_all(int a_timeout_ms);
// Возвращает: true если все завершены, false при таймауте
```

2.3.5.4 Конфигурация async мока Для включения async выполнения установите `.async = true` в конфигурации:

```c
// Async мок с задержкой
DAP_MOCK_DECLARE_CUSTOM(dap_client_http_request, {
    .enabled = true,
    .async = true,  // Выполнять callback асинхронно
    .delay = {
        .type = DAP_MOCK_DELAY_FIXED,
        .fixed_us = 50000  // 50ms
    }
});

// Mock обертка (выполняется асинхронно если был вызван dap_mock_async_init())
DAP_MOCK_WRAPPER_CUSTOM(void, dap_client_http_request,
    PARAM(const char*, a_url),
    PARAM(callback_t, a_callback),
    PARAM(void*, a_arg)
) {
    // Этот код выполняется в worker потоке после задержки
    a_callback("response data", 200, a_arg);
}
```

2.3.5.5 Утилиты

```c
// Получить количество pending задач
size_t dap_mock_async_get_pending_count(void);

// Получить количество completed задач
size_t dap_mock_async_get_completed_count(void);

// Выполнить все pending задачи немедленно ("промотать время")
void dap_mock_async_flush(void);

// Сбросить статистику
void dap_mock_async_reset_stats(void);

// Установить дефолтную задержку для async моков
void dap_mock_async_set_default_delay(uint32_t a_delay_ms);
```

2.3.5.6 Паттерн использования

```c
void test_async_http(void) {
    // Примечание: Ручная инициализация не нужна! Async система авто-инициализир

    volatile bool done = false;

    // Вызвать функцию с async моком (сконфигурированным с .async = true)
    dap_client_http_request("http://test.com", callback, &done);

    // Ждать async завершения
    DAP_TEST_WAIT_UNTIL(done, 5000, "HTTP request");

    // Или ждать все async моки
    bool completed = dap_mock_async_wait_all(5000);
    assert(completed && done);

    // Очистка (опционально, обрабатывается dap_mock_deinit())
    // dap_mock_deinit();  // Также авто-очищает async систему
}
```

Примечание: Async система автоматически инициализируется при старте mock фреймворка (через конструктор). Ручной `dap_mock_async_init()` нужен только если хотите настроить количество worker потоков.

2.4 4. Полные примеры

2.4.1 4.1 Тест стейт-машины (Пример из реального проекта)

Пример из `cellframe-srv-vpn-client/tests/unit/test_vpn_state_handlers.c`:

```c
#include "dap_test.h"
#include "dap_mock.h"
#include "vpn_state_machine.h"
#include "vpn_state_handlers_internal.h"

#define LOG_TAG "test_vpn_state_handlers"

// Объявление моков с простой конфигурацией
DAP_MOCK_DECLARE(dap_net_tun_deinit);
DAP_MOCK_DECLARE(dap_chain_node_client_close_mt);
DAP_MOCK_DECLARE(vpn_wallet_close);

// Мок с конфигурацией возвращаемого значения
DAP_MOCK_DECLARE(dap_chain_node_client_connect_mt, {
    .return_value.l = 0xDEADBEEF
});

static vpn_sm_t *s_test_sm = NULL;

static void setup_test(void) {
    // Примечание: dap_mock_init() вызывается авто, здесь не нужен
    s_test_sm = vpn_sm_init();
    assert(s_test_sm != NULL);
}

static void teardown_test(void) {
    if (s_test_sm) {
        vpn_sm_deinit(s_test_sm);
        s_test_sm = NULL;
    }
    // Опционально: dap_mock_deinit() для сброса моков между тестами
}

void test_state_disconnected_cleanup(void) {
    log_it(L_INFO, "ТЕСТ: state_disconnected_entry() очистка");

    setup_test();

    // Настройка состояния с ресурсами
    s_test_sm->tun_handle = (void*)0x12345678;
    s_test_sm->wallet = (void*)0xABCDEF00;
    s_test_sm->node_client = (void*)0x22222222;

    // Включение моков
    DAP_MOCK_ENABLE(dap_net_tun_deinit);
    DAP_MOCK_ENABLE(vpn_wallet_close);
    DAP_MOCK_ENABLE(dap_chain_node_client_close_mt);
```

```
    // Вызов обработчика состояния
    state_disconnected_entry(s_test_sm);

    // Проверка выполнения очистки
    assert(DAP_MOCK_GET_CALL_COUNT(dap_net_tun_deinit) == 1);
    assert(DAP_MOCK_GET_CALL_COUNT(vpn_wallet_close) == 1);
    assert(DAP_MOCK_GET_CALL_COUNT(dap_chain_node_client_close_mt) == 1);

    teardown_test();
    log_it(L_INFO, "[+] УСПЕХ");
}

int main() {
    dap_common_init("test_vpn_state_handlers", NULL);

    test_state_disconnected_cleanup();

    log_it(L_INFO, "Все тесты ПРОЙДЕНЫ [OK]");
    dap_common_deinit();
    return 0;
}
```

2.4.2 4.2 Мок с callback

```
#include "dap_mock.h"

DAP_MOCK_DECLARE(dap_hash_fast, {.return_value.i = 0}, {
    if (a_arg_count >= 2) {
        uint8_t *data = (uint8_t*)a_args[0];
        size_t size = (size_t)a_args[1];
        uint32_t hash = 0;
        for (size_t i = 0; i < size; i++) {
            hash += data[i];
        }
        return (void*)(intptr_t)hash;
    }
    return (void*)0;
});

void test_hash() {
    uint8_t data[] = {1, 2, 3};
    uint32_t hash = dap_hash_fast(data, 3);
    assert(hash == 6);  // Callback суммирует байты
}
```

2.4.3 4.3 Мок с задержками выполнения

Пример из dap-sdk/net/client/test/test_http_client_mocks.h:

```
#include "dap_mock.h"
```

```
// Мок с задержкой variance: симулирует реалистичные колебания сети
// 100мс ± 50мс = диапазон 50-150мс
#define HTTP_CLIENT_MOCK_CONFIG_WITH_DELAY ((dap_mock_config_t){ \
    .enabled = true, \
    .delay = { \
        .type = DAP_MOCK_DELAY_VARIANCE, \
        .variance = { \
            .center_us = 100000,   /* центр 100мс */ \
            .variance_us = 50000   /* разброс ±50мс */ \
        } \
    } \
})

// Объявление мока с симуляцией сетевой задержки
DAP_MOCK_DECLARE_CUSTOM(dap_client_http_request_full,
                        HTTP_CLIENT_MOCK_CONFIG_WITH_DELAY);

// Мок без задержки для операций очистки (мгновенное выполнение)
DAP_MOCK_DECLARE_CUSTOM(dap_client_http_close_unsafe, {
    .enabled = true,
    .delay = {.type = DAP_MOCK_DELAY_NONE}
});
```

2.4.4 4.4 Пользовательская линкер-обертка (Продвинутый уровень)

Пример из `test_http_client_mocks.c` с использованием `DAP_MOCK_WRAPPER_CUSTOM`:

```
#include "dap_mock.h"
#include "dap_mock_linker_wrapper.h"
#include "dap_client_http.h"

// Объявление мока (регистрация во фреймворке)
DAP_MOCK_DECLARE_CUSTOM(dap_client_http_request_async,
                        HTTP_CLIENT_MOCK_CONFIG_WITH_DELAY);

// Реализация пользовательской обертки с полным контролем
// DAP_MOCK_WRAPPER_CUSTOM генерирует:
// - сигнатуру функции __wrap_dap_client_http_request_async
// - массив void* args для фреймворка моков
// - Автоматическое выполнение задержки
// - Запись вызова
DAP_MOCK_WRAPPER_CUSTOM(void, dap_client_http_request_async,
    PARAM(dap_worker_t*, a_worker),
    PARAM(const char*, a_uplink_addr),
    PARAM(uint16_t, a_uplink_port),
    PARAM(const char*, a_method),
    PARAM(const char*, a_path),
    PARAM(dap_client_http_callback_full_t, a_response_callback),
    PARAM(dap_client_http_callback_error_t, a_error_callback),
    PARAM(void*, a_callbacks_arg)
) {
    // Пользовательская логика мока - симуляция асинхронного HTTP поведения
```

```
    // Это напрямую вызывает callback'и на основе конфигурации мока

    if (g_mock_http_response.should_fail && a_error_callback) {
        // Симуляция ошибочного ответа
        a_error_callback(g_mock_http_response.error_code, a_callbacks_arg);
    } else if (a_response_callback) {
        // Симуляция успешного ответа с настроенными данными
        a_response_callback(
            g_mock_http_response.body,
            g_mock_http_response.body_size,
            g_mock_http_response.headers,
            a_callbacks_arg,
            g_mock_http_response.status_code
        );
    }
    // Примечание: настроенная задержка выполняется автоматически перед этим код
}
```

CMakeLists.txt:

```
# Подключение auto-wrap помощника
include(${CMAKE_SOURCE_DIR}/dap-sdk/test-framework/mocks/DAPMockAutoWrap.cmake)

add_executable(test_http_client
    test_http_client_mocks.c
    test_http_client_mocks.h
    test_main.c
)

target_link_libraries(test_http_client
    dap_test     # Тест-фреймворк с моками
    dap_core     # Библиотека DAP core
    pthread      # Поддержка многопоточности
)

# Автогенерация --wrap флагов линкера сканированием всех исходников
dap_mock_autowrap(test_http_client)
```

2.4.5 4.5 Динамическое поведение мока

```
// Мок, который меняет поведение на основе счетчика вызовов
// Симулирует нестабильную сеть: ошибка 2 раза, затем успех
DAP_MOCK_DECLARE(flaky_network_send, {.return_value.i = 0}, {
    int call_count = DAP_MOCK_GET_CALL_COUNT(flaky_network_send);

    // Ошибка в первых 2 вызовах (симуляция сетевых проблем)
    if (call_count < 2) {
        log_it(L_DEBUG, "Симуляция сетевого сбоя (попытка %d)", call_count + 1);
        return (void*)(intptr_t)-1;  // Код ошибки
    }

    // Успех с 3-го и последующих вызовов
```

```
    log_it(L_DEBUG, "Сетевой вызов успешен");
    return (void*)(intptr_t)0;  // Код успеха
});

void test_retry_logic() {
    // Тест функции с повторными попытками при ошибке
    int result = send_with_retry(data, 3);  // Максимум 3 попытки

    // Должен завершиться успешно на 3-й попытке
    assert(result == 0);
    assert(DAP_MOCK_GET_CALL_COUNT(flaky_network_send) == 3);

    log_it(L_INFO, "[+] Логика повторных попыток работает корректно");
}
```

2.4.6 4.6 Мокирование в статических библиотеках

Пример теста, который мокирует функции внутри статической библиотеки dap_stream:

CMakeLists.txt:

```
include(${CMAKE_SOURCE_DIR}/dap-sdk/test-framework/mocks/DAPMockAutoWrap.cmake)

add_executable(test_stream_mocks
    test_stream_mocks.c
    test_stream_mocks_wrappers.c
)

target_link_libraries(test_stream_mocks
    dap_test
    dap_stream       # Статическая библиотека - функции внутри нужно мокировать
    dap_net
    dap_core
    pthread
)

target_include_directories(test_stream_mocks PRIVATE
    ${CMAKE_SOURCE_DIR}/dap-sdk/test-framework
    ${CMAKE_SOURCE_DIR}/dap-sdk/core/include
)

# Шаг 1: Автогенерация --wrap флагов из исходников теста
dap_mock_autowrap(test_stream_mocks)

# Шаг 2: Оборачивание статической библиотеки --whole-archive
# Это заставляет линкер включить все символы из dap_stream,
# включая внутренние функции, которые нужно мокировать
dap_mock_autowrap_with_static(test_stream_mocks dap_stream)
```

test_stream_mocks.c:

```
#include "dap_test.h"
```

```c
#include "dap_mock.h"
#include "dap_stream.h"
#include "dap_common.h"
#include <assert.h>

#define LOG_TAG "test_stream_mocks"

// Мокируем функцию, которая используется внутри dap_stream
DAP_MOCK_DECLARE(dap_net_tun_write, {
    .return_value.i = 0,  // Успешная запись
    .delay = {
        .type = DAP_MOCK_DELAY_FIXED,
        .fixed_us = 10000  // 10ms задержка
    }
});

// Оборачиваем функцию для мокирования
DAP_MOCK_WRAPPER_CUSTOM(int, dap_net_tun_write,
    PARAM(int, a_fd),
    PARAM(const void*, a_buf),
    PARAM(size_t, a_len)
) {
    // Логика мока - симулируем успешную запись
    log_it(L_DEBUG, "Mock: dap_net_tun_write called (fd=%d, len=%zu)", a_fd, a_l
    return 0;
}

void test_stream_write_with_mock(void) {
    log_it(L_INFO, "TEST: Stream write with mocked tun_write");

    // Создаём стрим (dap_stream использует dap_net_tun_write внутри)
    dap_stream_t *stream = dap_stream_create(...);
    assert(stream != NULL);

    // Выполняем запись - должна использовать мок dap_net_tun_write
    int result = dap_stream_write(stream, "test data", 9);

    // Проверяем что мок был вызван
    assert(result == 0);
    assert(DAP_MOCK_GET_CALL_COUNT(dap_net_tun_write) > 0);

    dap_stream_delete(stream);
    log_it(L_INFO, "[+] Test passed");
}

int main() {
    dap_common_init("test_stream_mocks", NULL);

    test_stream_write_with_mock();

    dap_common_deinit();
```

```
    return 0;
}
```

Ключевые моменты: 1. dap_mock_autowrap() должно быть вызвано **до** dap_mock_autowrap_with_static() 2. Укажите все статические библиотеки, в которых нужно мокировать функции 3. --whole-archive может увеличить размер исполняемого файла 4. Работает только с GCC, Clang и MinGW

2.4.7 4.7 Асинхронное выполнение моков

Пример демонстрации async mock callback'ов с thread pool:

```
#include "dap_mock.h"
#include "dap_mock_async.h"
#include "dap_test_async.h"

// Async мок для HTTP запроса с задержкой 50ms
DAP_MOCK_DECLARE_CUSTOM(dap_client_http_request, {
    .enabled = true,
    .async = true,  // Выполнять в worker потоке
    .delay = {
        .type = DAP_MOCK_DELAY_FIXED,
        .fixed_us = 50000  // 50ms реалистичная сетевая латентность
    }
});

// Mock обертка - выполняется асинхронно
DAP_MOCK_WRAPPER_CUSTOM(int, dap_client_http_request,
    PARAM(const char*, a_url),
    PARAM(http_callback_t, a_callback),
    PARAM(void*, a_arg)
) {
    // Этот код выполняется в worker потоке после задержки 50ms
    const char *response = "{\"status\":\"ok\",\"data\":\"test\"}";
    a_callback(response, 200, a_arg);
    return 0;
}

static volatile bool s_callback_executed = false;
static volatile int s_http_status = 0;

static void http_response_callback(const char *body, int status, void *arg) {
    s_http_status = status;
    s_callback_executed = true;
    log_it(L_INFO, "HTTP ответ получен: status=%d", status);
}

void test_async_http_request(void) {
    log_it(L_INFO, "TEST: Async HTTP request");

    // Инициализировать async mock систему с 1 worker потоком
    dap_mock_async_init(1);
```

```
    s_callback_executed = false;
    s_http_status = 0;

    // Вызвать HTTP запрос - мок выполнится асинхронно
    int result = dap_client_http_request(
        "http://test.com/api",
        http_response_callback,
        NULL
    );

    assert(result == 0);
    log_it(L_DEBUG, "HTTP запрос инициирован, ждём callback...");

    // Ждать завершения async мока (до 5 секунд)
    DAP_TEST_WAIT_UNTIL(s_callback_executed, 5000, "HTTP callback");

    // Проверка
    assert(s_callback_executed);
    assert(s_http_status == 200);

    // Альтернатива: ждать все async моки
    bool all_completed = dap_mock_async_wait_all(5000);
    assert(all_completed);

    log_it(L_INFO, "[+] Async mock тест пройден");

    // Очистка async системы
    dap_mock_async_deinit();
}

// Пример fast-forward: тест без реальных задержек
void test_async_with_flush(void) {
    dap_mock_async_init(1);

    s_callback_executed = false;

    // Запланировать async задачу с большой задержкой
    dap_client_http_request("http://test.com", http_response_callback, NULL);

    // Вместо ожидания 50ms, выполнить немедленно
    dap_mock_async_flush();  // "Промотать" время

    // Callback уже выполнен
    assert(s_callback_executed);

    log_it(L_INFO, "[+] Fast-forward тест пройден");
    dap_mock_async_deinit();
}
```

Преимущества Async Моков: - Реалистичная симуляция сетевой/IO латентности - Не требуется полная инфраструктура dap_events в unit тестах - По-

токобезопасное выполнение - Детерминированное тестирование с `flush()` - Отслеживание статистики с `get_pending_count()` / `get_completed_count()`

2.5 5. Глоссарий

Асинхронная операция - Операция, завершающаяся в непредсказуемое будущее время

Auto-Wrapper - Система авто-генерации флагов линкера `--wrap` из исходников

Callback - Указатель на функцию, выполняемую при событии

Condition Polling - Повторная проверка условия до выполнения или таймаута

Condition Variable - pthread примитив для синхронизации потоков

Constructor Attribute - GCC атрибут для запуска функции до main()

Designated Initializers - C99 инициализация: `{.field = value}`

Global Timeout - Ограничение времени для всего набора тестов через SIGALRM

Linker Wrapping - `--wrap=func` перенаправляет вызовы в `__wrap_func`

Mock - Фальшивая реализация функции для тестирования

Monotonic Clock - Источник времени, не зависящий от системных часов

Poll Interval - Время между проверками условия

pthread - Библиотека POSIX threads

Return Value Union - Объединение для типобезопасных возвратов моков

Self-Test - Тест, проверяющий сам фреймворк тестирования

Thread Pool - Набор worker потоков для выполнения async задач

Thread-Safe - Корректно работает при конкурентном доступе

Timeout - Максимальное время ожидания

Union - C тип, хранящий разные типы в одной памяти

2.6 6. Решение проблем

2.6.1 Проблема: Тест зависает бесконечно

Симптом: Тест выполняется бесконечно без завершения
Причина: Асинхронная операция никогда не сигнализирует о завершении
Решение: Добавьте защиту глобальным таймаутом

```
dap_test_global_timeout_t timeout;
if (dap_test_set_global_timeout(&timeout, 30, "Tests")) {
    log_it(L_ERROR, "Test timeout!");
}
```

Профилактика: Всегда используйте DAP_TEST_WAIT_UNTIL с разумным таймаутом

2.6.2 Проблема: Высокая загрузка CPU

Симптом: 100% CPU во время теста
Решение: Увеличьте интервал polling или используйте pthread helpers

```
cfg.poll_interval_ms = 500;  // Менее частый polling
```

2.6.3 Проблема: Мок не вызывается (выполняется реальная функция)

Симптом: Вызывается реальная функция вместо мока
Причина: Отсутствует флаг линкера `--wrap`
Решение: Проверьте конфигурацию CMake и флаги линкера

```
# Проверьте наличие флагов линкера
make VERBOSE=1 | grep -- "--wrap"

# Должно быть: -Wl,--wrap=function_name
```

Исправление: Убедитесь что `dap_mock_autowrap(target)` вызван после `add_executable()`

2.6.4 Проблема: Неправильное возвращаемое значение

Симптом: Мок возвращает неожиданное значение
Решение: Используйте правильное поле union

```
.return_value.i = 42      // int
.return_value.l = 0xDEAD  // указатель
.return_value.ptr = ptr   // void*
```

2.6.5 Проблема: Нестабильные тесты (периодические сбои)

Симптом: Иногда проходят, иногда падают
Причина: Состояние гонки, недостаточные таймауты или предположения о времени
Решение: Увеличьте таймауты и добавьте толерантность для проверок времени

```
// Для сетевых операций - используйте щедрый таймаут
cfg.timeout_ms = 60000;  // 60 сек для сетевых операций

// Для проверок времени - используйте диапазон толерантности
uint64_t elapsed = measure_time();
assert(elapsed >= 90 && elapsed <= 150);  // ±50мс толерантность

// Используйте вариативную задержку для реалистичной симуляции
DAP_MOCK_SET_DELAY_VARIANCE(func, 100000, 50000);  // 100мс ± 50мс
```

2.6.6 Проблема: Ошибка компиляции “undefined reference to __wrap”

Симптом: Ошибка линкера о __wrap_function_name
Решение: Убедитесь что dap_mock_autowrap() вызван в CMakeLists.txt

```
include(${CMAKE_SOURCE_DIR}/dap-sdk/test-framework/mocks/DAPMockAutoWrap.cmake)
dap_mock_autowrap(my_test)
```

2.6.7 Проблема: Callback мока не выполняется

Симптом: Мок возвращает настроенное значение, но логика callback не выполняется
Причина: Callback не зарегистрирован или мок отключен
Решение: Проверьте что callback установлен и мок включен

```
// Объявите с inline callback (предпочтительно)
DAP_MOCK_DECLARE(func_name, {.enabled = true}, {
    // Ваша логика callback здесь
    return (void*)42;
});

// Или установите callback в runtime
DAP_MOCK_SET_CALLBACK(func_name, my_callback, user_data);

// Убедитесь что мок включен
DAP_MOCK_ENABLE(func_name);
```

Примечание: Возвращаемое значение callback переопределяет конфигурацию .return_value

2.6.8 Проблема: Мок не работает для функций в статической библиотеке

Симптом: Функции из статической библиотеки (lib*.a) не мокируются, вызывается реальная функция
Причина: Линкер исключает неиспользуемые символы из статических библиотек, поэтому --wrap не применяется
Решение: Используйте dap_mock_autowrap_with_static() для оборачивания статической библиотеки флагами --whole-archive

```
# После обычной линковки и dap_mock_autowrap()
dap_mock_autowrap(test_target)
```

```
# Оборачиваем статическую библиотеку --whole-archive
dap_mock_autowrap_with_static(test_target dap_http_server)
```

Проверка:

```
make VERBOSE=1 | grep -E "--whole-archive|dap_http_server"
# Должно быть: -Wl,--whole-archive ... dap_http_server ... -Wl,--no-whole-archiv
```

Важно: Порядок важен! Сначала `dap_mock_autowrap()`, затем `dap_mock_autowrap_with_st`

2.6.9 Проблема: Ошибка линкера “multiple definition”

Симптом: Ошибка `multiple definition of 'function_name'` при использовании `--whole-archive`
Причина: Некоторые символы определены в нескольких библиотеках
Решение: `dap_mock_autowrap_with_static()` автоматически добавляет `--allow-multiple-definition`, но если проблема сохраняется:

```
# Явно добавьте флаг
target_link_options(test_target PRIVATE "-Wl,--allow-multiple-definition")
```

Альтернатива: Используйте `--whole-archive` только для конкретных библиотек, которые требуют мокирования

2.6.10 Проблема: Задержка не работает

Симптом: Мок выполняется мгновенно несмотря на конфигурацию задержки
Решение: Проверьте что задержка установлена после объявления мока

```
DAP_MOCK_DECLARE(func_name);
DAP_MOCK_SET_DELAY_MS(func_name, 100);  // Установка после объявления
```