DAP SDK Test Framework - Полное Руководство

Асинхронное тестирование, моки и автоматизация тестов

Команда разработки Cellframe

27 октября 2025

Содержание

1 Информация о документе

Версия: 1.0.0
Дата: 27 октября 2025
Статус: Production Ready
Язык: Русский

1.1 История изменений

Версия	Дата	Изменения	Автор
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

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 V4** - Мокирование функций без модификации кода
- **Auto-Wrapper System** - Автоматическая конфигурация линкера
- **Self-Tests** - 21 тест, валидирующий 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 - □ Защиту от зависаний (глобальный + для каждой операции) - □ Эффективное ожидание (polling + condition variables) - □ Изоляцию зависимостей (мокирование) - □ Реалистичную симуляцию (задержки, ошибки) - □ Потокобезопасные операции - □ 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
Задержки	Fixed, Range, Variance	Реалистичная симуляция
Callbacks	Inline + Runtime	Динамическое поведение моков
Auto-Wrapper	Bash/PowerShell скрипты	Автоматическая настройка
Self-Tests	21 комплексный тест	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

```c
// 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)

```c
#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:

```
target_link_libraries(my_test dap_test dap_core pthread)
```

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
```

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

2.3.2.1 Объявление

```
DAP_MOCK_DECLARE(func_name);
DAP_MOCK_DECLARE(func_name, {.return_value.i = 42});
DAP_MOCK_DECLARE(func_name, {.return_value.i = 0}, { /* callback */ });
```

2.3.2.2 Макросы управления

```
DAP_MOCK_ENABLE(func_name)          // Включить мок
DAP_MOCK_DISABLE(func_name)         // Выключить мок
```

```
DAP_MOCK_RESET(func_name)           // Сбросить состояние
DAP_MOCK_SET_RETURN(func_name, value)  // Установить возвращаемое значение
DAP_MOCK_GET_CALL_COUNT(func_name)     // Получить счётчик вызовов
```

2.3.2.3 Конфигурация задержек

```
DAP_MOCK_SET_DELAY_FIXED(func_name, microseconds)  // Фиксированная задержка
DAP_MOCK_SET_DELAY_FIXED_MS(func_name, milliseconds)  // В миллисекундах
DAP_MOCK_SET_DELAY_RANGE(func_name, min_us, max_us)  // Диапазон
DAP_MOCK_SET_DELAY_VARIANCE(func_name, center_us, variance_us)  // Разброс
DAP_MOCK_CLEAR_DELAY(func_name)  // Очистить задержку
```

2.4 4. Примеры использования

2.4.1 4.1 Тест стейт-машины

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

#define LOG_TAG "test_vpn_sm"
#define TIMEOUT_SEC 30

bool check_connected(void *data) {
    return vpn_sm_get_state((vpn_sm_t*)data) == VPN_STATE_CONNECTED;
}

void test_connection() {
    vpn_sm_t *sm = vpn_sm_init();
    vpn_sm_transition(sm, VPN_EVENT_USER_CONNECT);

    dap_test_async_config_t cfg = DAP_TEST_ASYNC_CONFIG_DEFAULT;
    cfg.timeout_ms = 10000;
    cfg.operation_name = "VPN connection";

    bool ok = dap_test_wait_condition(check_connected, sm, &cfg);
    dap_assert_PIF(ok, "Should connect within 10 sec");

    vpn_sm_deinit(sm);
}

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

    dap_test_global_timeout_t timeout;
    if (dap_test_set_global_timeout(&timeout, TIMEOUT_SEC, "VPN Tests")) {
        return 1;
    }

    test_connection();

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

2.4.2 4.2 Мок с callback

```
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
}
```

2.5 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-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

Симптом: Тест выполняется бесконечно
Решение: Добавьте глобальный таймаут

```
dap_test_set_global_timeout(&timeout, 30, "Tests");
```

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

Симптом: Выполняется реальная функция
Решение: Проверьте флаги линкера

```
make VERBOSE=1 | grep -- "--wrap"
```

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

Симптом: Иногда проходят, иногда падают
Решение: Увеличьте таймаут, добавьте допуск

```
cfg.timeout_ms = 60000;  // 60 сек для сети
assert(elapsed >= 90 && elapsed <= 150);  // ±50мс допуск
```