Разработка системы для отладки ядра операционной системы

Автор: Александр Романович Поляков, 471 группа
Научный руководитель: ст. преп. Я. А. Кириленко
Рецензент: старший менеджер группы программных инструментов разработки ООО "Синопсис СПб" А. С. Колесов

Санкт-Петербургский государственный университет
Кафедра системного программирования

11 июня 2019г.

Введение

- Операционная система (ОС) — распространённое и критичное к ошибкам программное обеспечение (ПО)
- Объём кода ядра ОС — от тысяч до десятков миллионов строк
- Отладка — средство локализации ошибок в программном коде
- Современные ОС оперируют схожими понятиями

Постановка задачи

Цель: разработка расширяемой системы для отладки ядра операционной системы

Задачи:

- Выполнить обзор существующих решений
- Разработать архитектуру системы
- Реализовать требуемую системы
- Провести апробацию системы
  - добавить поддержку выбранной целевой архитектуры
  - добавить поддержку выбранной ОС
  - провести тестовую отладочную сессию

Отладка ядра ОС

Применимость

- Отладка функциональности ядра ОС
- Отладка пользовательских приложений, находящихся в режиме ядра
- Отладка *unikernel* приложений

Обзор: основные подходы

- Отладочный вывод
- Модификация исходного кода ядра для добавления возможности трассировки его состояния
- Использование отладчика

Обзор: использование отладчика

- Прикладная программа (API ОС, например *ptrace*)
- Bare-metal приложение
  - Управление состоянием процессора
  - Количество потоков, видимых отладчиком, ограничено
- Операционная система

Обзор: использование отладчика

	Открытость кода	**Расширяемость мн-ва ОС**	**Расширяемость мн-ва архитектур**
ARM DS-5	—	+	—
WinDbg	—	—	+
GDB	+	+/—	+
pyocd	+	+	—
XNU lldbmacros	+	—	+

Таблица: Базовые параметры существующих решений

Обзор: использование отладчика

GDB

- Сфокусирован на ядре *Linux*
- Добавление ОС требует модификации исходного кода отладчика

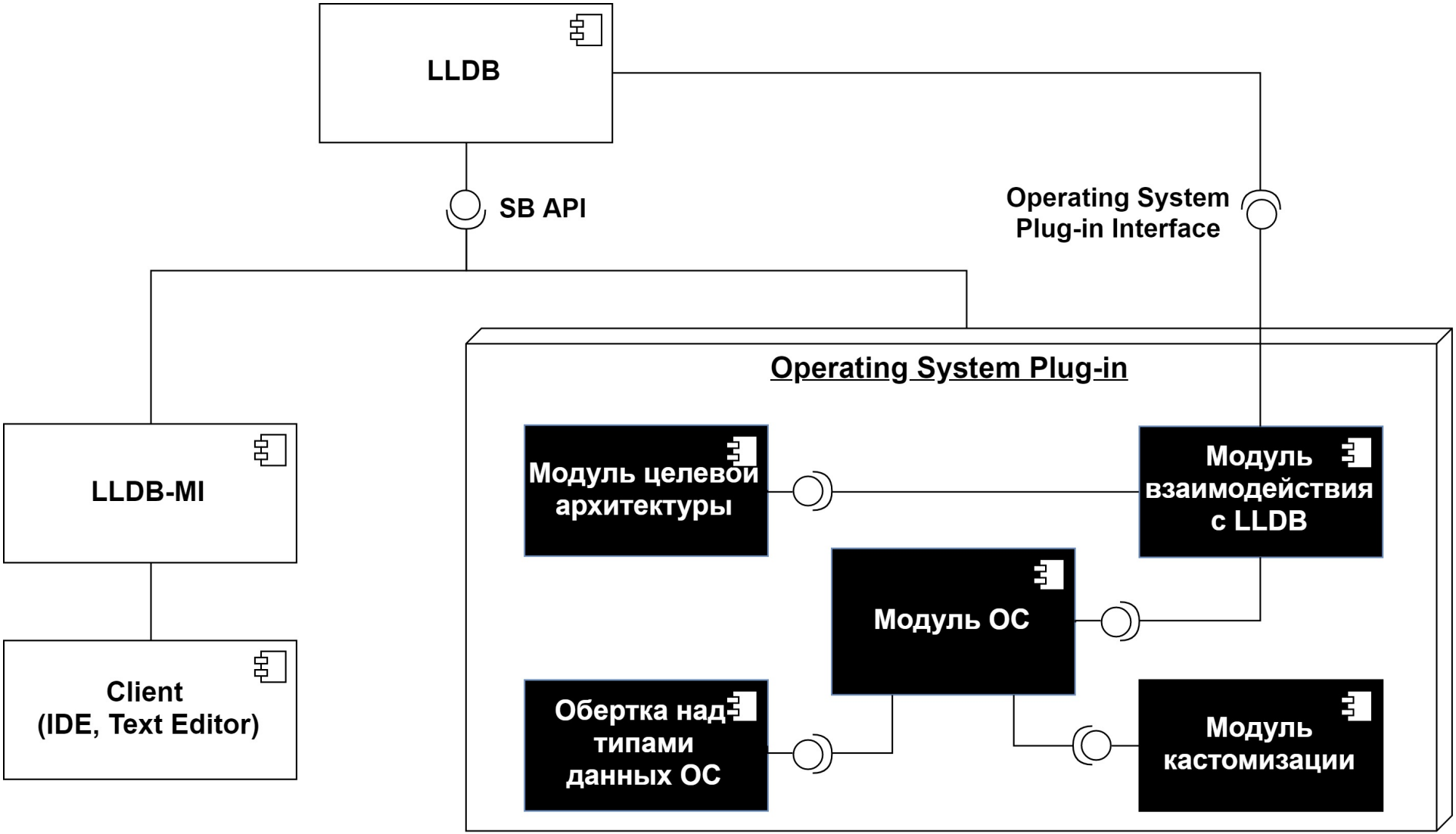
pyocd

- Поддерживается только архитектура *ARM Cortex-M*
- Низкий уровень абстракции

XNU lldbmacros

- Переиспользует функциональность отладчика *LLDB*
- Высокий уровень абстракции
- Разработан для отладки *XNU* — ядро *macOS*, *iOS*

Архитектура системы



Модули системы

Модуль взаимодействия с LLDB

- Инициализация плагина
- Предобработка информации
- Инкапсулирование работы с отладчиком

Модуль целевой архитектуры

- Описание набора регистров

Модуль ОС

- Интерфейс взаимодействия плагина с ОС
- Сущности, общие для различных ОС

Модули системы

Обёртка над типами данных ОС

- Взаимодействие с объектами ОС как с объектами языка Python
- Повышение уровня абстракции

Модуль кастомизации

- Переопределение способа представления объектов ОС в виде строки
- Добавление консольных команд отладчика

Модуль трассировки и журналирования

- Отладка плагина

Особенности реализации

- Расширяемость
- Отсутствие зависимостей между отладчиком и ОС
- Возможность кастомизации
- Обёртка над типами языка реализации ОС
  - Поддерживаются: C, C++, Objective-C, Objective-C++, Swift
- Трассировка и журналирование
- Документация
- Аннотации типов (Python)

Апробация

- Целевая архитектура — *ARC*
- Операционная система — *Zephyr*
  - Программные потоки
  - Восстановление контекста неактивных потоков
- Три потока в ядре ОС
- Одноядерный режим процессора

Апробация

```
(lldb) thread info all
thread #1: tid = 0x0001, 0x000004bc zephyr.elf`
...
```

Листинг 1: Фрагмент отладочной сессии без использования плагина

```
(lldb) thread info all
thread #1: tid = 0x80000068, 0x000004bc zephyr.elf`
...
thread #2: tid = 0x80000000, 0x000004b2 zephyr.elf`
...
thread #3: tid = 0x800000e8, 0x00000c10 zephyr.elf`
...
```

Листинг 2: Фрагмент отладочной сессии с использованием плагина

Заключение

- Выполнен обзор существующих решений в области отладки ОС
  - использование отладочного вывода
  - трассировка
  - использование отладчика
- Разработана архитектура системы
- Реализована требуемая система
- Проведена апробация системы
  - добавлена поддержка целевой архитектуры *ARC*
  - добавлена поддержка операционной системы *Zephyr*
  - проведены тестовые отладочные сессии

Результаты работы внедрены в разработки компании *Synopsys*

```
(lldb) thread info all
thread #1: tid = 0x0001, 0x000004bc zephyr.elf`
helloLoop(my_name="threadB", my_sem=0x800022c4,
other_sem=0x800022b4) at main.c:46,
stop reason = breakpoint 1.1
```

Листинг 3: Фрагмент отладочной сессии без использования плагина

```
(lldb) thread info all
thread #1: tid = 0x80000068, 0x000004bc zephyr.elf`
helloLoop(my_name="threadB", my_sem=0x800022c4,
other_sem=0x800022b4) at main.c:46, name = 'thread_b'

thread #1: tid = 0x80000000, 0x000004b2 zephyr.elf`
helloLoop [inlined] k_current_get at kernel.h:29,
name = 'thread_a'

thread #1: tid = 0x800000e8, 0x00000c10 zephyr.elf`
k_cpu_idle + 16, name = 'idle'
```

Листинг 4: Фрагмент отладочной сессии с использованием плагина

```
# struct Point {
#   int x;
#   int y;
# };

# Global variable defined somewhere
# in the debugging program
point = FindGlobalVariable(target, 'point')

# Without module
a = point.GetChildMemberWithName('x').\
    GetValueAsSigned() + 1

# With module
a = point.x + 1
```

Листинг 5: Обёртка над типами языка реализации ОС