

# Динамический анализ ARINC-653 совместимой ОСРВ с помощью LLVM

В.Ю. Чешцов

cheptsov@ispras.ru

А.В. Хорошилов

khoroshilov@ispras.ru

Ivannikov Institute for System Programming  
of the Russian Academy of Sciences

# ОСРВ JetOS

Разрабатывается в ИСП РАН совместно с ГосНИИАС.

Стадия прототипа. 40000 строк кода базовой системы (C99).

Широкий функционал:

- ARINC 653
- C Standard Library (подмн.)
- OpenGL SC 2.0  
(ИПИМ им. М.В.Келдыша РАН)
- Embedded C++
- Ada Standard Library (подмн.)

Кроссплатформенность:

- PowerPC 32-bit (e500v2, e500mc)
- ARMv7 (Cortex-A7, Cortex-A9)
- MIPS (MIPS32r1)
- X86 (i486-совместимые)
- В планах 64-битные архитектуры

# Постановка задачи

1. В составе инструментальных средств ОСРВ необходима поддержка современных способов динамического анализа функционального ПО.
2. Требуется упростить процесс обнаружения и локализации ошибок в прототипе ОСРВ.

# Санитайзеры LLVM

LLVM — набор технологий для компиляции и сборки кода.

LLVM Sanitizers — набор инструментов от Google для динамического анализа кода, встраиваемый в программу автоматически на этапе компиляции.

—AddressSanitizer

—EfficiencySanitizer

—LeakSanitizer

—libFuzzer

—MemorySanitizer

—SafeStack

—ThreadSanitizer

—UndefinedBehaviorSanitizer

# Требования к инструментам

- Кроссплатформенность, нетребовательность к ресурсам, простота в интеграции, высокая производительность.
- Снижение усилий при отладке проблемы путём явного указания на место ошибки до того, как её последствия проявятся.
- Обнаружение скрытых ошибок, которые никак себя не проявляют при заданной конфигурации на тестовых наборах.

*AddressSanitizer + MemorySanitizer + UndefinedBehaviourSanitizer*

# Альтернативные решения

Коммерческие: Insure++, PurifyPlus, Intel Inspector, WinDBG

Открытые: Valgrind, Dr.Memory

- Узкий функционал
- Сложная адаптируемость
- Специализация под платформу
- Высокая цена (за продукт или его доработку)

# Элементы санитайзера

1. Поддержка на уровне компилятора
2. Поддержка на уровне среды выполнения
  - Адаптация встроенной
  - Адаптация сторонней
  - Разработка собственной

# Проблемы адаптации

- Совместимость инструмента с конфигурацией целевой платформы при сборке.
- Необходимость обеспечения работы компонента до завершения инициализации инструмента.
- Отсутствие достаточной интерфейсной поддержки со стороны ОС для портирования среды исполнения.
- Растущее потребление памяти и варьирующееся снижение производительности.
- Требования по аннотации и изменению кода.



# Особенности инициализации

Требуется выделение части ОС, которая работает в процессе инициализации среды исполнения инструмента.

- Через глобальные конструкторы (с приоритетом 0)
- Раздельная компиляция (init code, memcpu, memset...)
- Информация о раскладке памяти (стеки, кучи, сегменты)
- Аннотация ассемблерных конструкций (syscall)
- Thread Local Storage и стандартная библиотека C

# Накладные расходы и реальное время

Неравномерное замедление в 2–3 раза, чувствительное к использованию оптимизаций. Цепочки инструментов и опций делают замедление плохо предсказуемым.

В числе решений:

- Локальная корректировка таймаутов
- Глобальное замедление таймера в ядре

# Потребление RAM и ROM

- Объём исполняемого файла растёт на 50–100%
- Теневая память требует 15–100% адресуемой памяти
- Формула вычисления теневого адреса фиксирована
- Последовательность адресов упрощает реализацию
- Раскладку памяти можно передать изнутри или снаружи
- Инструмент может мониторить всю память ФПО.

# Ручное аннотирование

Увеличение эффективности инструмента за счёт вставки специальных конструкций в код. Например, в функции на основе ранее сформулированных требований.

- Кодогенерация на основе шаблонов
- Проксирование части функций
- Препроцессирование кода на основе точек привязки

# Найденные дефекты

- Порядка 10 реальных ошибок в отдельных компонентах прототипа разной степени критичности.
- Инфраструктурные проблемы:
  - В раскладке памяти
  - В системе сборки и конфигурирования
  - Неучтённые требования инструментов
- Ошибки работы с памятью в нескольких тестах из тестового набора, проверяющего соответствие ARINC-653.
  - Включая неактуальные проверки спецификации.

# Дальнейшие планы

- Повысить степень интеграции с нашей экосистемой и портировать другие инструменты.
- Продолжить работу над адаптацией фазз-тестирования в совокупности с санитайзерами LLVM в рамках ОСРВ.
- Рассмотреть идеи улучшения самих инструментов, включая добавление новых проверок.

# TODO

- Ручные проверки памяти в Asan и Msan не имеют API проверок с учётом выравнивания структур.
- Убрать вызовы libc из встроенной среды исполнения некоторых инструментов.
- Реализовать проверки использования volatile при обращении к регионам ММО памяти.
- Обеспечить детерминизм в замедлении инструмента без внесения изменений в код.
- Доработать реализацию поддержки SPE.

Спасибо за внимание!

Вопросы?