Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.Б07-мм

Динамическая бинарная трансляция в RISC-V с помощью Instrew

Михайлов Илья Игоревич

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
ассистент кафедры системного программирования, Косарев Д. С.

Санкт-Петербург
2024

Оглавление

Введение

Бинарная трансляция — форма трансляции, где на вход подается последовательность инструкций одной архитектуры процессора, а на выходе получаем инструкции другой архитектуры. Такая трансляция может осуществляться с целью эмулировать какой-либо набор инструкций, чтобы запускать программы, скомпилированные не под назначенную архитектуру, либо же скомпилированные под другое расширение набора команд (важный пример — расширения RISC-V ISA). Примерами эмуляторов являются: QEMU[1], Rosetta 2, Box64[2]. Также исходные инструкции транслировать в инструкции той же архитектуры с помощью инструментов бинарного анализа для выполнения оптимизаций и внесения патчей, например, когда рекомпиляция/декомпиляция осложнена; для профиляции и отладки транслируемой программы. Примерами таких инструментов являются Valgrind[3], Pin, Dynamo.

Бинарную трансляцию можно классифицировать по способу выполнения на статическую и динамическую. Также бинарные трансляторы могут использовать поднятие инструкций до некоторого более высокоуровневого промежуточного представления. QEMU использует TCG, Valgrind использует VEX. Но хочется использовать более распространенное промежуточное представление, позволяющее делать качественные оптимизации. Одним из таких является LLVM IR.

Хочется иметь динамический бинарный транслятор с поднятием в LLVM IR у которого есть поддержка архитектуры RISC-V, как хоста.

[1] https://www.qemu.org/
[2] https://box86.org/
[3] https://valgrind.org/

1. Постановка задачи

Целью работы является добавление поддержки архитектуры RISC-V для ДБТ Instrew. Для ее выполнения были поставлены следующие задачи:

1. выполнить обзор архитектуры Instrew;
2. реализовать минимальную функциональность для запуска Instrew на RISC-V;
3. протестировать Instrew на простых программах.

2. Обзор

Для обзора предлагается рассмотреть популярные эмуляторы и инструменты для бинарного анализа, а также отдельно рассмотреть динамические бинарные трансляторы с поднятием инструкций в LLVM IR. В данной работе не будет рассмотрен динамический транслятор Rosetta 2, так как не имеет поддержки RISC-V и не является программным обеспечением с открытым исходным кодом, несмотря на то, что существуют попытки реверс-инжиниринга[4].

2.1. Динамические бинарные трансляторы

2.1.1. Эмуляторы

QEMU — динамический бинарный транслятор с открытым исходным кодом, предназначенный для эмуляции обширного числа архитектур, который также поддерживает аппаратную виртуализацию. В качестве промежуточного представления используется TCG, в которое поднимаются базовые блоки транслируемой программы[1]. Транслятор, исходный код и транслируемый код находятся в одном адресном пространстве во время исполнения[2]. Поддерживает RISC-V как хост-архитектуру и как целевую архитектуру.

Box64 — динамический бинарный транслятор с открытым исходным кодом, позволяющий запускать x86/x86_64 программы на архитектурах ARM и RISC-V[5]. Использует JIT-компилятор dynarec, который транслирует базовые блоки в хост архитектуру[6]. Замеры с помощью nbench показали, что box64 работает быстрее QEMU[7].

2.1.2. Инструменты для бинарного анализа

Valgrind — инструмент для динамического бинарного анализа с открытым исходным кодом и большим числом плагинов/динамических

[4] https://ffri.github.io/ProjectChampollion/
[5] https://box86.org/2024/08/box64-and-risc-v-in-2024/
[6] https://box86.org/2024/07/revisiting-the-dynarec/
[7] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10077985

анализаторов, такие как: Memcheck, Callgrind, Cachegrind. В качестве промежуточного представления используется VEX, в которое поднимаются суперблоки транслируемой программы, где суперблок — последовательность инструкций, в котором может быть несколько точек выхода[4]. Транслятор, плагин и транслируемая программа находятся в одном адресном пространстве. Поддерживает архитектуру RISC-V.

2.2. Динамические бинарные трансляторы с поднятием в LLVM IR

2.2.1. Instrew

Instrew — динамический бинарный транслятор с поднятием инструкций в LLVM IR, открытым исходным кодом и клиент-сервер архитектурой. Как хост архитектуру процессора поддерживает x86_64 и Aarch64, как целевую — x86_64, Aarch64, RISC-V. С помощью бинарного лифтера Rellume поднимает функции в LLVM IR. Может использоваться как эмулятор, так и как инструмент для бинарного анализа кода[2].

2.2.2. DBILL

DBILL — инструмент для динамического бинарного анализа, использующий TCG и LLVM IR, как промежуточные представления. Благодаря фронтэнду QEMU и тулчейну LLVM поддерживается множество хост и целевых архитектур[8]. Из-за двух промежуточных представлений и поднятия в TCG только базовых блоков, есть проблемы с производительностью. Исходный код не был найден.

2.3. Выводы

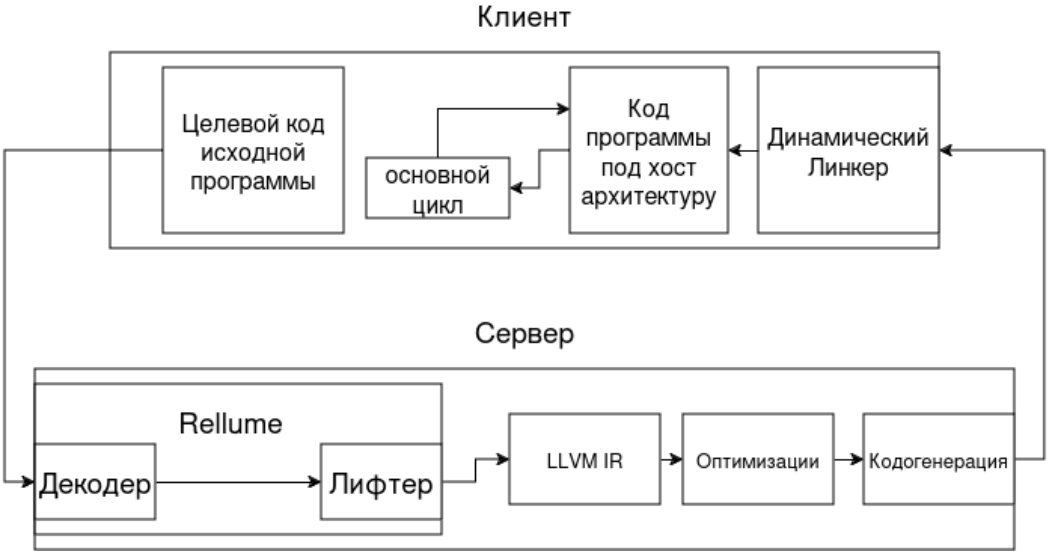
По результатам замеров[3], Instrew оказался более производительным, чем QEMU и Valgrind. Хоть Box64 тоже оказывается производительнее чем QEMU, но не имеет API для произведения динамического

[8] https://doi.org/10.1145/2674025.2576213

бинарного анализа, что в сравнении с Instrew делает его менее универсальным. Далее данная работа будет сосредоточена именно на ДБТ Instrew с поднятием в LLVM IR и добавлении поддержки архитектуры RISC-V для такого транслятора.

3. Обзор Instrew

3.1. Архитектура



Instrew реализует клиент-сервер архитектуру:

- клиент, написанный на языке C и ассемблере, и выполняющий функции:
  - загрузки целевой программы в адресное пространство;
  - эмуляции состояния процессора;
  - отправки функций целевой программы серверу;
  - динамической линковки ELF объектов, полученных от сервера;
  - загрузка, разрешение символов и проведение релокаций для полученного ELF объекта;
  - эмуляция системных вызовов;
  - вызов функции, полученной после линковки ELF объекта;
  - кэширование;
- сервер, написанный на языке C++, выполняющий функции:
  - декодирование и поднятие функции целевой программы в LLVM IR с помощью Rellume;

– оптимизация LLVM IR;

– кодогенерация;

– создание ELF объекта;

4. Реализация поддержки клиента Instrew

Листинг 1: Релокация

```
uint64_t syma = (uintptr_t) sym + patch_data->addend;
uint64_t pc = patch_data->patch_addr;
int64_t prel_syma = syma - (int64_t) pc;

case R_RISCV_32_PCREL:

if (!rtld_elf_signed_range(prel_syma, 32, "R_RISCV_32_PCREL"))
   return -EINVAL;
rtld_blend(tgt, 0xffffffffff, prel_syma);
break;
```

Листинг 2: Заголовок

```
ASM_BLOCK(
    .text;
    .global _start;
    .type _start, %function;
_start:
    .weak __global_pointer;
    .hidden __global_pointer;
    .option push;
    .option norelax;
    lla gp, __global_pointer;
    .option pop;
    mv a0, sp;
    .weak _DYNAMIC;
    .hidden _DYNAMIC;
    lla a1, _DYNAMIC;
    andi sp, sp, -16;
    tail __start_main;
);
```

Листинг 3: Системный вызов

```
static size_t syscall2(int n, size_t a, size_t b)
{
    register size_t a7 __asm__("a7") = n;
    register size_t a0 __asm__("a0") = a;
    register size_t a1 __asm__("a1") = b;
    __asm_syscall("r"(a7), "0"(a0), "r"(a1))
}
```

5. Тестирование

- Удалось успешно протестировать:
  - программах без libc, исполняющих системные вызовы;
  - факториале и фибоначчи, выводящих результат в exit code;
  - факториале с минимальным рантаймом, который выводит результат на экран.
- Не удалось протестировать на:
  - динамически слинкованных программах и программах использующих libc;
  - программах, с флагами сборки -pie;
  - программах, где конкретная релокация еще не реализована.
- Вывод:
  - работают статически слинкованные программы, с простым рантаймом и системными вызовами

Заключение

В результате работы были выполнены следующие задачи:

1. выполнен обзор архитектуры Instrew;
2. реализованы процессорно-специфические патчи и проставлены константы;
3. реализована эмуляция системных вызовов, PLT трамплины и минимальный набор ELF релокаций;
4. Instrew был протестирован на статически слинкованных программах, с простым рантаймом.

Код доступен по данной ссылке.

Список литературы

[1] Bellard Fabrice. QEMU, a fast and portable dynamic translator // Proceedings of the Annual Conference on USENIX Annual Technical Conference. — ATEC '05. — USA : USENIX Association, 2005. — P. 41.

[2] Engelke Alexis, Schulz Martin. Instrew: leveraging LLVM for high performance dynamic binary instrumentation // Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. — VEE '20. — New York, NY, USA : Association for Computing Machinery, 2020. — P. 172–184. — URL: https://doi.org/10.1145/3381052.3381319.

[3] Engelke Alexis Friedrich. Optimizing Performance Using Dynamic Code Generation : Ph. D. thesis / Alexis Friedrich Engelke ; Technische Universität München. — 2021. — P. 163. — URL: https://mediatum.ub.tum.de/1614897.

[4] Nethercote Nicholas, Seward Julian. Valgrind: a framework for heavy-weight dynamic binary instrumentation // Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation. — PLDI '07. — New York, NY, USA : Association for Computing Machinery, 2007. — P. 89–100. — URL: https://doi.org/10.1145/1250734.1250746.