开题报告

计算机学院 2023 级数硕士 9 班 3220231370 傅泽

1. 选题依据

1.1. 选题背景及意义

随着中美科技战持续升温，针对传统的闭源 CPU 指令集架构和操作系统等软件的禁令成为了中国发展软硬件技术产业的极大桎梏。为最小化这些禁令的影响，越来越多的国内信息技术公司正在将软硬件逐步迁移至独立可控的开源平台上。这其中，精简指令集 RISC-V 及程序设计语言 rust 分别以其开源免费、内存安全性高等优点赢得了无数软件开发者的青睐，越来越多开发者在二者的基础上开发了诸多操作系统及配套软件。

然而，考虑到 RISC-V 架构问世时间较短，基于 RISC-V 架构开发的操作系统尚未形成如运行在 X86 架构上的操作系统那般完善的软件生态，将运行在 X86 架构上的软件进行必要的修改，进而移植到 RISC-V 架构上的操作系统就成为了拓展 RISC-V 操作系统软件生态的重要一步。这种跨架构的移植的流程较为复杂，需要考虑到移植过程中的各种因素，特别是运行时库、系统调用等在源架构和目标架构上的不同点。现代软件的依赖项繁多，各依赖项之间的依赖关系错综复杂，且均有可能需要不同的系统调用等作为支撑，因此移植工作能够进行的必要条件之一，就是将所有依赖项可能使用到的系统调用全部实现。

不难发现，上述以依赖项为最小单位，以实现所有系统调用为基础的跨架构软件移植流程存在诸多不足，其中最为关键的不足有以下几点：

1. 代码冗余。若深入研究待移植软件与其依赖项的关系即可发现，待移植软件可能仅使用了其依赖项中的部分功能，而未使用的功能虽然对软件工作而言完全多余，但仍然会被编译并成为最终获得的二进制可执行文件中的一部分。从源代码角度分析，这种代码冗余为软件开发者从逻辑上分析待移植软件的工作原理造成了干扰，从而增加了移植工作的难度；从编译所得角度分析，这种代码冗余增加了不必要的编译时间，和不必要的可执行文件体积。
2. 系统调用实现冗余。正如前文所言，以依赖项做最小单位，为其实现全部所需的系统调用，也将为那些实际上并未被使用的依赖项内容实现系统调用。而实际上这些并未被使用的依赖项内容在移植过程中完全可以忽略，此时为这些内容实现的系统调用就成为了冗余，加重了移植工作的无意义工作量。

若能摒弃以依赖项为最小单元，转而采用以函数调用为最小单元的分析方法，则可以有效避免上述不足。由于采用函数调用为最小分析单元，原本的依赖关系可以拆分为依赖项和软件之间的函数间依赖关系。通过分析这种关系，可以识别出依赖项中未被使用的函数，进而给出裁剪建议。软件开发者可以凭借这些建议移除冗余代码。一言以蔽之，以函数调用为最小单元的分析方法在大大减少移植工作的难度、减少需要实现的系统调用数量的同时，对于帮助软件开发者理清程序逻辑，精简代码与编译所得文件之体积等方面亦有积极作用。

本课题提出开发一款针对 rust 程序设计语言的优化工具，其能够以函数调用作为最小单元对指定的 rust 项目（即 rust crate）进行分析，识别其中及其依赖项中的冗余代码，为开发者给出代码裁剪建议。为证明该工具确实能有效辅助进行跨架构软件移植，本课题将以 Substrate 区块链的节点模板为例，使用该工具进行分析，统计裁剪前后代码的体积变化，并与未裁剪的原始代码进行比较，证明该工具的有效性。

1.2. 国内外研究现状

1.2.1. rust 的中间表示

编译型程序设计语言的编译工具链在将源代码编译为目标平台的二进制可执行文件时，往往会选择先将源代码编译为某种形式的中间表示（IR），再将中间表示编译为目标平台的机器代码。在部分语言中，为方便从不同的角度进行全面完善的检查，一些语言

还将使用数种不同层级的中间表示，待上层级中间表示通过检查后，再将其编译为下一层级的中间表示，运行下一步的检查，如此逐层降低层级，直至获得最后的机器码。这些中间表示从不同角度呈现了不同的诊断信息，有利于软件开发者开发外部工具对其进行诸如静态检查等操作。

rust 程序设计语言亦采用了这种多层级中间表示的思路。截至目前，rust 一共使用了如下四种中间表示：

1. 高层级中间表示（HIR）：HIR 是 rust 编译器 rustc 中最高层级的中间表示，由对源代码进行语法解析、宏展开等处理之后的抽象语法树转换而来。其形式和 rust 源代码尚有相似之处，但将一些语法糖展开为了更易于分析的形式，例如 `for` 循环将被展开为 `loop` 循环等。
2. 带类型的高层级中间表示（THIR）：该中间表示是由 HIR 在完成类型检查后降低层级而来，主要用于枚举穷尽性检查、不安全行为检查和下一层中间表示的构造。和 HIR 相比，THIR 最大的不同在于其内容仅保留了源代码中可执行代码的部分，因此诸如 `struct` 和 `trait` 等结构将不会在 THIR 中出现，从而非常适合用于分析 rust crate 中的函数定义信息。
3. 中层级中间表示（MIR）：这种中间表示于 RFC 1211 中初次引入[1]，用于控制流相关的安全检查，例如借用检查器。它进一步将一些语法糖展开，引入了在 rust 源代码中不可能出现的语句，同时也会执行控制流分析。由于 rustc 提供了一组虽不稳定但切实可用的应用编程接口用于和 MIR 交互，这使得 MIR 成为了诸多外部工具处理 rust 程序代码的不二选择，如 MIRChecker、Kani Analyzer 等均采用 MIR 作为其分析对象。
4. LLVM 中间表示（LLVM IR）：该中间表示由 MIR 降低层级而来，是 LLVM 编译工具链可以识别并加以利用的中间表示。然而，由于 LLVM IR 的表达能力有限，许多 rust 源代码中隐含的信息无法合理表示，故如今基于它进行进一步处理的分析工具越来越少。

[1] **https://blog.rust-lang.org/2016/04/19/MIR.html**

1.2.2. PRAZI

为提升代码可复用性，许多现代编程语言如 JavaScript、rust 等均提供了公开可用的软件包托管平台，并辅之以清单文件来描述一个软件包依赖于其他软件包的情况。若将软件包甲依赖软件包乙的这种依赖关系视为一条有向边，那么在软件托管平台上存在的所有软件包之间即能构造出一张错综复杂的包依赖关系网络（Package Dependency Network，PDN）。通过研究包依赖关系网络，可以获得软件包级别的依赖信息，例如某个包依赖了哪些其他包的具体版本，哪些包被依赖次数最多等信息。

近年来，随着一些托管平台上出现投毒行为，所有依赖于这些软件包的软件均面临小至程序输出异常，大至重要资料被窃取、计算机系统遭到破坏等风险。

1.2.3. MIRAI

MIRAI 是工作在 rust 编程语言的中层中间表达（MIR）上的分析工具[2]。

2. 研究内容

3. 研究方案

4. 预期研究成果

5. 本课题创新点

参考文献

[2] **https://github.com/facebookexperimental/MIRAI**