我国基础软件现状，以编译器为例

编译原理第一次作业

卢雨轩 19071125

2022 年 2 月 21 日

基础软件包括什么？回答这个问题，只需要看计算机科学与技术专业的本科生学的四大基础课程：《计算机组成原理》、《数据库原理》、《操作系统》、《编译原理》。在计算机相关的任何应用都无法脱离这四门基础课程的知识，同理，计算机在工程领域的软件中，EDA 软件、数据库、操作系统以及编译器是四种最基础的基础软件。下面，本文以世界以及中国的编译器软件为例讨论我国基础软件现状。

国际上，知名的编译器有 GNU （Gnu is Not Unix) 组织的『GCC （the GNU Compiler Collection)』是 Linux 中最知名的编译器，也是世界范围内应用最广泛的编译器，支持 C、C++、Objective-C、Fortran 等多种语言。由 UIUC 研发，并被苹果公司广泛应用的『LLVM 编译基础设施（Low-Level Virtual Machine Compiler Infrastructure）』通过分离编译器前后端并通过 LLVM 这一中间语言桥接，实现了多种语言与 ISA 支持，并提供强大的优化能力。除了上述两个开源的编译器之外，也有大量闭源的编译器被广泛应用：由 Intel 公司开发的『ICC （Intel C++ Compiler)』主要面向 Intel 平台，由于其出色的定向（面向 Intel x86 和 amd64 架构 CPU）的优化能力以及向量化（vectorization）能力，被广泛应用在 HPC 场景。由微软公司开发的『MSVC（MicroSoft Virtual C++)』主要面向 Win/x86 场景，随后适配 amd64 以及 arm32、arm64 架构，作为 Win 32 开发的主要编译器，也得到了广泛的应用。

更现代的语言在实现能够自举的编译器时，主要有两种思路。第一种是如 Go 语言，从 0 开始实现编译器，没有任何依赖，甚至不依赖于系统汇编器，而是自己实现了一套全新的汇编器。第二种是如 Rust 语言，实现面向 LLVM 中间语言的编译器前端，使得可以直接适配 LLVM 所支持的所有 ISA，并获得其优秀的优化能力。也正因如此，rustc 的工程量十分小，但是其输出的 LLVM 的质量显著低，使得在编译一个同样的程序时，其相比与 GCC 系列编译器的速度慢 5-10 倍[1]。同时，一些新的语言也同时采用两种方式，如微软公司的 TypeScript 语言，可以通过其编译器配合巴别塔（the BabelJS）转译为 JavaScript，也可以通过另一个编译器编译为运行在嵌入式设备上的二进制代码。这俩类思路各有利弊，前者工程量更大，需要自己支持各类 ISA 以及各类优化，但是更为自由。后者程序产物需要依赖现有工具链，但是工程量更小，开发更为便捷。

现有的『国产』编译器中，『太极（Taichi）』[2]收获了很多的关注。作为一门领域特定的编程语言（Domain Specific Programming Language），其具有优秀的 GPU 开发能力，相较 CUDA 等传统库，更容易上手、使用。另一个获得广泛关注乃至争议的语言是『木兰』。木兰作为一款面向嵌入式平台的小型编译器，由于在宣传上的失误以及广大群众对于『中芯』等项目的恐惧，木兰受到了很大的批评以及指责。笔者个人认为，单从技术上来看，在面向嵌入式平台的同时提供基于 Python 的『套皮』模拟器，是十分正常且合理的，并不能因其使用了 Python 而受到批评。

由此可见，对于基础语言（如 C++、C）的编译器，我国还是显著依赖于世界范围内的开源编译器，并没有实现真正的『自主』。那么，下一个需要讨论的问题就是使用开源编译器『安全』吗？如果不经过任何代码审计就拿来使用，显然是不安全的。无论是在编译的生成结果中嵌入恶意代码，还是在标准库中故意暴露漏洞，都有可能对软件造成很大的破坏。比如，对于安全要求较高的地方，如 OpenSSL 等密码学库，都不依赖于标准库提供的随机数生成器，而是选择自己实现随机数生成器（密码学的大部分算法都要求随机数具有足够的熵。如果随机数不够强，攻击者就可以找到攻击向量）。

但是，对于一个平台，除非从硬件到固件到软件都是自主设计或者经过安全审计的，否则总有一些环节需要『信任』。如，在上述的例子中，OpenSSL 选择不信任编译器标准库提供的随机数（定然，也一定

程度上因为功能性和强度的需求），选择自己实现随机数生成器。但是，在生成随机数的算法依赖与操作系统提供的真随机数作为种子。凭什么说操作系统提供的真随机数种子是可以信任的呢？如果使用在线服务生成真随机数（如 random.org 声称提供基于大气噪音生成的真随机数），又凭什么说这些服务是可以信任的呢？如果依赖 CPU 硬件提供的、基于电气热噪声的随机数，也同样存在这个信任问题。对于开源操作系统提供的基于用户输入的真随机数，可以通过审计操作系统源码的方式来确保真实性，但是其他随机数生成方式仍然存在信任问题。因此，大公司往往采用自主设计的随机数生成硬件来采样随机数，避免暴露攻击向量给攻击者，如 Cloudflare 公司提供 SSL 服务，需要大量的、具有足够熵的随机数，因此其采用自主设计的『熔岩灯随机数（LavaRand）』[3] 硬件生成随机数。

『生成可信的随机数』这个小问题都有如此复杂的信任问题，更不要说复杂问题以及完整的工程的信任问题了。如，现有的操作系统提供的对于应用程序安全性全部依赖于 CPU 硬件提供的内核态与应用态的隔离，但是如果 CPU 本身是不可信的呢？Intel x86 前日爆出一个隐藏的，用于修改微码的指令 [4]。该指令虽然对于现有系统的安全性没有危害，但是不禁引人思考：如果 CPU 中有一个隐藏的指令，可以不经授权直接从用户态提升到内核态，那么现代操作系统提供的所有安全保证都会像气球一样，一触即溃。

既然如此，那么，是否有必要投入编译器等基础软件的开发呢？经过以上的讨论，答案显然是『有』。对于已有成熟开源产品的应用场景，如编译器或操作系统，我们可以采取审计源码的方式，维护一个我们可控的分支。对于没有成熟开源产品的领域，如 EDA 软件，我们则要投入精力开发。面向消费领域，可能实现很高的安全性没有很大的意义，但是对于国防领域和科研领域，安全性则是重中之重。但是，实现自主的这条路任重而道远。我们通过自主实现编译器、汇编器、链接器，实现了编译软件的可控，接下来要面对的就是操作系统的安全问题；我们通过编写操作系统或者审计操作系统源码来保证操作系统的安全性，接下来要面对的就是 CPU 的安全问题；我们通过自主实现 CPU 来保证安全性，接下来还要面对外设的安全性问题：如果网卡中有后门怎么办？由此可见，在追求安全、可控的道路上没有终点。现有来看，基础软件中，开源编译器、操作系统、数据库等不存在所谓『卡脖子』问题，但是如 EDA 等工程领域的软件，我们仍然受制于人（如中芯和海思等，被列入实体清单后无法使用 EDA 软件）。但是，尽管过程艰难，最终的目标仍然是有必要的，需要坚持的走下去。

笔者一直以来关注国产软件和硬件的开发，见证了汉芯、龙芯、木兰、太极等项目的兴衰，一直想总结一下个人对于这方面的观点。这次借助作业这个机会，有感而发，陈述一下自己的观点，不知不觉已经写了这么多。请老师同学们批评指正！

有关参考文献：本文撰写过程中大部分资料均来自笔者的记忆与积累，无法找到准确的出处和参考文献。部分在编写过程中搜索的资料陈列如下：

参考文献

[1] Why does rust compile a simple program 5-10 times slower than gcc/clang? - stack overflow[EB/OL]. https://stackoverflow.com/questions/37362640/why-does-rust-compile-a-simple-program-5-10-times-slower-than-gcc-clang/37365065.

[2] HU Y. The taichi programming language[C/OL]//SIGGRAPH '20: ACM SIGGRAPH 2020 Courses. Association for Computing Machinery, 2020: 1–50. https://doi.org/10.1145/3388769.3407493.

[3] Randomness 101: Lavarand in production[EB/OL]. https://blog.cloudflare.com/randomness-101-lavarand-in-production/.

[4] Undocumented x86 instructions in intel cpus that can modify microcode | hacker news[EB/OL]. https://news.ycombinator.com/item?id=26519693.