Porting the Steel Bank Common Lisp Compiler and Runtime to the Nintendo Switch NX Platform

Charles Zhang

Yukari "Shinmera" Hafner

charleszhang99@yahoo.com

shinmera@tymoon.eu

Shirakumo.org

Zürich, Switzerland

ABSTRACT

The Nintendo Switch (NX) is a 64-bit ARM-based platform for video games with a proprietary micro-kernel operating system. Notably this system does not give programs the ability to mark pages as executable or give access to thread signal handlers, both of which present a significant hurdle to SBCL's intended bootstrap process and runtime operation. We present our efforts to adapt the SBCL runtime and compiler to deploy applications onto the NX platform.

CCS CONCEPTS

• **Software and its engineering** → **Runtime environments**; **Dynamic compilers**; *Garbage collection*; Software creation and management.

KEYWORDS

Common Lisp, SBCL, porting, ARM, aarch64, NX, Experience Report

ACM Reference Format:

Charles Zhang and Yukari "Shinmera" Hafner. 2025. Porting the Steel Bank Common Lisp Compiler and Runtime to the Nintendo Switch NX Platform. In *Proceedings of the 18th European Lisp Symposium (ELS'25)*. ACM, New York, NY, USA, 2 pages.

1 INTRODUCTION

The Nintendo Switch (codename NX) is a handheld video game console based on the ARM 4 Cortex-A57 64-bit architecture[4]. It features a proprietary micro-kernel operating system called the "Nintendo Switch system software"[6] (NX OS), and normally only runs software that has been licensed, approved, and digitally signed and encrypted by Nintendo.

ELS'25, May 19–20 2025, Zürich, Switzerland

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM

Developing licensed software for the NX has to be done via Nintendo's own proprietary SDK, which they only distribute under a non-disclosure agreement (NDA). An open-source third-party alternative to the SDK is available[1], but this cannot be used for licensed software.

The OS-provided runtime environment is only directly suitable for C and C++ software. Other game engines such as Unity, Unreal, and Godot do provide exporting functionality to the NX, meaning that ports for the runtime environments they rely on such as C#, Lua, and GDScript have been developed, but remain closed-source.

A port of a C-based runtime for Common Lisp such as ECL[2] is conceivable. However, due to the high performance requirements of video games and the relatively low-power platform of the NX, we deemed it unlikely for such a port to lead to a useful platform. Besides, relying on a C runtime would not eliminate all the difficulties we've encountered in porting SBCL anyway.

Particularly, every known Common Lisp implementation compiles a part of its runtime from Lisp source code incrementally into itself, and all user-code is also incrementally compiled into the Lisp image, as opposed to being batch-compiled and linked like other languages such as C. We cannot incrementally compile code on the NX due to platform security restrictions. The consequence is that all implementation and all user code needs to be compiled ahead of time. We discuss the intricacies of this in section 3 and section 4.

The NX OS further does not provide user signal handlers. This is a problem for the Garbage Collector (GC), as SBCL relies on inter-thread signalling to park threads during garbage collection. While SBCL does provide a safepoints mechanism for GC that is used on Windows, this mechanism had not been tested on any other platform, and as-is still did not meet the requirements of this locked-down platform. We discuss the GC in detail in section 5.

While we can now compile and deploy complex Common Lisp applications to the NX, some parts of the runtime remain unsupported, and we discuss our future efforts in this regard in section 7.

2 RELATED WORK

Rhodes[5] outlines the methodology behind the SBCL bootstrapping process.

A pure Common Lisp bootstrapping process as Durand et al.[3] outline would however not notably improve our process, as all our challenges arise from not being able to bootstrap on the desired target platform in the first place, and needing to handle the low-level system construction processes to be amenable for the NX' restrictions.

Citing information on the NX' operating system is difficult as it is a closed-source platform with all usual information placed under NDA. All publicly available information is from security research such as by Roussel-Tarbouriech et al.[6] and reverse-engineering[1].

Particularly, we are unaware of any publication about the porting of other runtime environments to the NX, such as C#, JavaScript, Lua, etc.

3 BUILD SYSTEM

4 RELOCATION

5 GARBAGE COLLECTION

6 CONCLUSION

7 FURTHER WORK

8 ACKNOWLEDGEMENTS

We would like to thank Douglas Katzman for his help and advice for various parts of the porting effort, as well as the rest of the SBCL maintenance team for their continuous improvements to the SBCL platform.

REFERENCES

[1] Switchbrew. URL https://switchbrew.org/.
[2] Guiseppe Attardi. The embeddable common lisp. In *Papers of the fourth international conference on LISP users and vendors*, pages 30–41, 1994.
[3] Irène A Durand and Robert Strandh. Bootstrapping common lisp using common lisp. In *EUROPEAN LISP SYMPOSIUM*, 2019.
[4] Thomas Morgan. New switch mod delivers real-time cpu, gpu and thermal monitoring – and the results are remarkable. *Eurogamer*, 2020.
[5] Christophe Rhodes. Sbcl: A sanely-bootstrappable common lisp. In *Workshop on Self-sustaining Systems*, pages 74–86. Springer, 2008.
[6] Gauvain Tanguy Henri Gabriel Roussel-Tarbouriech, Noel Menard, Tyler True, Tini Vi, et al. Methodically defeating nintendo switch security. *arXiv preprint arXiv:1905.07643*, 2019.