



SHIRAKUMO

Porting SBCL to the Nintendo Switch

Yukari Hafner, Charles Zhang

<https://shirakumo.org>

The Device



- CPU: ARM 4 Cortex-A57 64-bit
- OS: “Horizon OS”, proprietary micro-kernel
- SDK: C++, proprietary version of Clang



Immediate Challenges

- Everything is proprietary and under NDA
⇒ Scarce public information
- The OS is not BSD or even fully POSIX
⇒ Need new OS abstractions
- There are no inter-thread signals
⇒ Can't use usual GC tricks
- We are not allowed to create executable pages
⇒ No compilation at runtime



Basic Ideas

- Everything is proprietary and under NDA
 ⇒ Only publicise our own interfaces
- The OS is not BSD or even fully POSIX
 ⇒ Write C(++) shim libraries for access
- There are no inter-thread signals
 ⇒ Use safepoints
- We are not allowed to create executable pages
 ⇒ Compile everything on linux and shrinkwrap
- The OS enforces strict Address Space Layout Randomization (ASLR)
 ⇒ Make all code and data fully relocatable and position independent



History of the Project

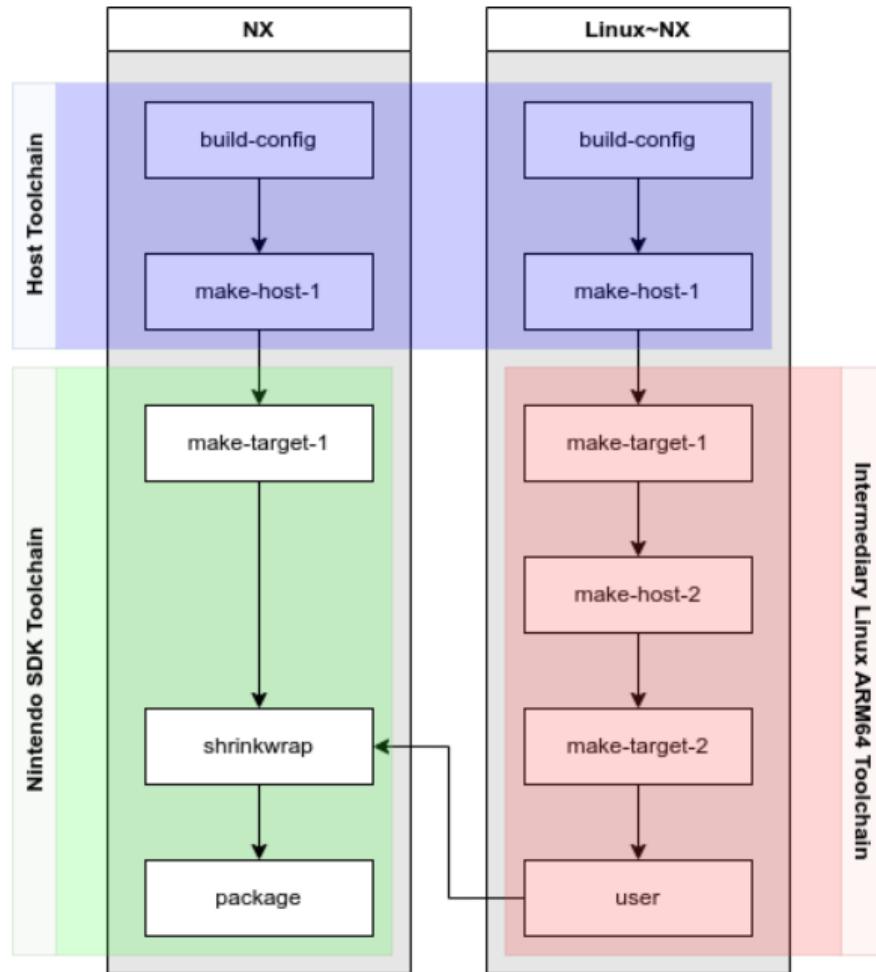
- Initial discussion on #sbcl and at ELS 2022 about porting SBCL to Nintendo Switch
- Decided to seriously pursue project at ELS 2023
⇒ Sketched out project plan in the airport afterwards
- Most of the initial work and design problems solved in the following months
- Debugging GC issues and improving debugging experience in 2024
- Game engine examples stabilized, Kandria running in early 2025



A Standard SBCL Build

- build-config
 - ⇒ Gather system info
- make-host-1
 - ⇒ Emit C headers and support files
- make-target-1
 - ⇒ Compile the C runtime on the target
- make-host-2
 - ⇒ Cross-compile the compiler on the host
- make-target-2
 - ⇒ Use the compiler from the host to incrementally compile the rest on the target





Intermediary Linux ARM64 Toolchain



Relocation

- Lisp objects are full of absolute pointers to other objects
- Lisp objects live in Lisp spaces mapped at fixed addresses
- Saving an image dumps Lisp spaces to disk verbatim
- Reloading an image is fast; just `mmap` the on-disk spaces into the process.
- ⇒ Problem for NX: enforced ASLR means no means to map spaces to desired addresses



Relocation (cont.)

- Solution: Relocate all Lisp spaces on start up
- ⇒ Problem for NX: Code generation hardwires some primitive object addresses.
- Solution: Change codegen to be position independent.
- Code objects in SBCL also contain Lisp pointers (constants)
- ⇒ Problem for NX: Relocation and moving GC needs to fix up these constants, but code objects are not writable
- Solution: Segregate code instructions and data in code objects into different ELF sections offline, rewrite code instructions to reference r/w section



Relocation example

```
(defun f ()  
  (cons '(42 . 1958) 'els2025))
```

A Lisp function referencing the constants '(42 . 1958) and
'ELS2025



Relocation example (cont.)

```
Dynamic space (as core image)
.....
0x7006B167A0: 00000000000000084 = 42
0x7006B167A8: 0000000000003912 = 1956
.....
0x7006B95890: 000000000003002D ; symbol header word
0x7006B95898: 4EF23449F9800000 = 284433385822765056
0x7006B958A0: 0000000000000009 = #<unbound>
0x7006B958A8: 0000000000000000 = 0
0x7006B958B0: 0000000300200117
0x7006B958B8: 0000007006B9587F = "ELS2025"
.....
0x70075CA890: 0000001A40000035 ; code object header word
0x70075CA898: 0000000000000030 = 24 ; boxed area size (bytes)
0x70075CA8A0: 0000000000000000 = 0 ; unused
0x70075CA8A8: 0000007006C44143 = #S(DEBUG-INFO ...) ; debug info
0x70075CA8B0: 0000007006B167A7 = '(42 . 1956)' ; constants[0]
0x70075CA8B8: 0000007006B9589F = 'ELS2025' ; constants[1]
0x70075CA8C0: 0000762D00000001 ; junk
0x70075CA8C8: 0000000300000130 = 6442451096 ; junk
.ENTRY F()
    STR LR, [CFP, #8]
```

Relocation example (cont.)

```
.text (in pie-shrinkwrap-sbcl.s)
    .word 0x1A4000035 ; header
    .word 0x30
    .word 0x0
    .word 0x0
    .word 0x0
    .word 0x0
    ...
.ENTRY F()
    STR LR, [CFP, #8]
    CBNZ NARGS, L2
    ADD CSP, CFP, #24
    LDR R0, [THREAD, #16]
    STR R0, [CFP, #16]
    STR WNULL, [THREAD, #40]
    LDP TMP, LR, [THREAD, #88]
    ADD R0, TMP, #16
    CMP R0, LR
    BHI L3
    STR R0, [THREAD, #88]
L0: ADD R0, TMP, #7
    LDR R2, &constants[1046]
    LDR R1, &constants[1047]
    STP R2, R1, [TMP]
    STR WZR, [THREAD, #40]
    LDR WLR, [THREAD, #44]
```

Dynamic space (in pie-shrinkwrap-sbcl.o)

```
.....
0x7006B167A0: 0000000000000084 = 42
0x7006B167A8: 0000000000003912 = 1956
.....
0x7006B95890: 000000000003002D ; symbol header word
0x7006B95898: 4EF23449F9800000 = 284433385822765056
0x7006B958A0: 0000000000000009 = #<unbound>
0x7006B958A8: 0000000000000000 = 0
0x7006B958B0: 0000000300200117
0x7006B958B8: 0000007006B9587F = "ELS2025"
```

Garbage Collection

- Garbage collection on SBCL on non-Windows platforms uses POSIX signals to stop-the-world.
- ⇒ Problem for NX: No POSIX signals.
- Solution: Use safepoints (implemented already on Windows)
- BUT: The safepoint mechanism still uses virtual memory hardware exceptions to stop the current thread
- ⇒ Problem for NX: No ability to handle operating system exceptions
- Solution: Change safepoint codegen to poll state flags explicitly



Misc Issues

- SBCL uses hardware exceptions to signal some common conditions
- ⇒ Solution: Change to full calls to ERROR and SIGNAL
- CLOS requires runtime compilation for dispatch JIT
- ⇒ Solution: Use an interpreter where absolutely necessary
- The existing shrinkwrap tool is not GC-safe for precise GC platforms like ARM64, random crashes with large cores
- ⇒ Solution: Complicate the build process even more by using a separate SBCL with memory spaces mapped away from the offline core's spaces to shrinkwrap



Live Demo



Further Work

- Optimising CLOS dispatch ahead of time
⇒ Christophe? 😕
- Optimising Trial and Kandria
⇒ Lots of profiling work that can be done on PC
- Porting to the Nintendo Switch 2
⇒ As soon as plebians like us get access from almighty Nintendo





Thank you!

Consider supporting our work:



<https://patreon.com/shinmera>