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

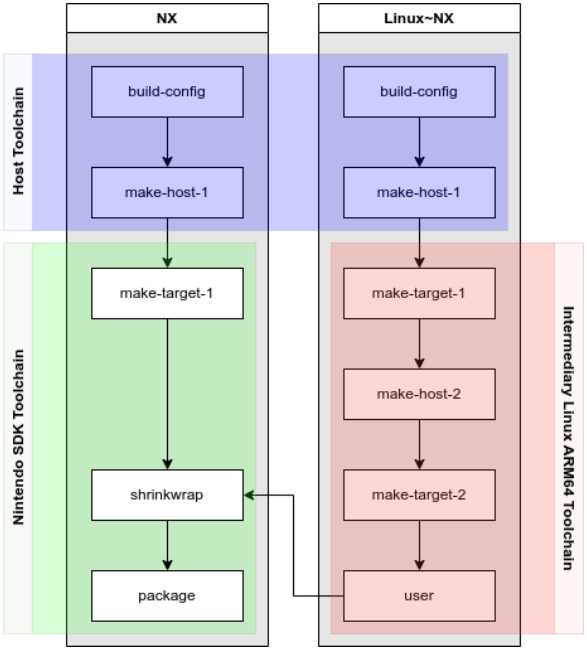
- Everything is proprietary and under NDA
  ⇒ Only publicise our own interfaces
- The OS is not BSD or even fully POSIX
  ⇒ Write C(++) shim libraries for access
- There are no inter-thread signals
  ⇒ Use safepoints
- We are not allowed to create executable pages
  ⇒ Compile everything on linux and 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
  $\Rightarrow$ 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


NX
Linux~NX
Host Toolchain
build-config
make-host-1
build-config
make-host-1
Nintendo SDK Toolchain
make-target-1
shrinkwrap
package
make-target-1
make-host-2
make-target-2
user
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.
- $\Rightarrow$ Problem for NX: enforced ASLR means no means to map spaces to desired addresses

Relocation (cont.)

- Solution: Relocate all Lisp spaces on start up (support existed for most Lisp spaces already, except for one)
- $\Rightarrow$ 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)
- $\Rightarrow$ 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`

Representation of the compiled code object (in boldface) for the function #'£ in memory, along with the cons and symbol objects it references.

```
.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]
    CBZ LR, L1
    BRK #9
L1: MOV CSP, CFP
    LDP CFP, LR, [CFP]
    CMP NULL, #0
    RET
L2: BRK #16
L3: MOVZ TMP, #16
    LDR R0, 0x70055CA8C8
    BLR R0
    B L0
```

```
.data (in pie-shrinkwrap-sbcl.s)
      ....
constants[1046]:
 .word 0x7006B167A7 ; '(42 . 1956)
constants[1047]:
 .word 0x7006B9589f ; 'ELS2025
```

```
Dynamic space (in pie-shrinkwrap-sbcl.o)
      ....
0x7006B167A0: 0000000000000084 = 42
0x7006B167A8: 0000000000003912 = 1956
      ....
0x7006B95890: 000000000003002D ; symbol header word
0x7006B95898: 4EF23449F9800000 = 2844333385822765056
0x7006B958A0: 0000000000000009 = #<unbound>
0x7006B958A8: 0000000000000000 = 0
0x7006B958B0: 0000000300200117
0x7006B958B8: 0000007006B9587F = "ELS2025"
```

Representation of the same code object and the cons and symbol objects it references after code/data segregation and the rest of the shrinkwrapping process.

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 (challenge: do this in a race-free way!)

Misc Issues

- SBCL uses hardware exceptions to signal some common conditions
- $\Rightarrow$ Solution: Change to full calls to `ERROR` and `SIGNAL`
- CLOS requires runtime compilation for dispatch JIT
- $\Rightarrow$ 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
- $\Rightarrow$ 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

SHIRAKUMO

Thank you!

Consider supporting our work:

https://patreon.com/shinmera