Porting the Swift 3.1 Programming Language to Haiku

Joseph Calvin Hill, (return0e)
calvin@hakobaito.co.uk
London, UK,
IRC: return0e, Trac: return_0e
Trac/Github tickets: (#1238, #1268 , #1269)
Github: @return

Q: Will you treat Google Summer of Code as full time employment?
A: I will treat GSoC as full time employment.

Q: How many hours per week will you work?
A: I am able to work 35 - 60 hours per week during the coding weeks.

Time allocations are as follows:

May 4th - June 15th	38 Hours pw
June 16th - July 16th	60 Hours pw
July 17th - August 1st	50 Hours pw
August 2nd - August 31st	55 Hours pw

List all obligations (and their dates) that may take time away from GSoC:

I am currently 8 months in on my 9 - 5 work placement (Ends in July 31st) and can take time off to work on GSoC full-time.

Q: Are you using Google Summer of Code to fulfil a university requirement?
A: No.

Estimated last day of classes/exams:
N/A No exams this year.

Estimated first day of classes:
October 3rd.

Introduction

Haiku supports various compiled and interpreted programming languages for developers to choose from, mostly ported via haikuporter recipes. Both POSIX and haikuporter have eased the porting process and addressed the lack of platform support for more sophisticated compilers and libraries such as GCC 4+, LLVM and Clang. Newer languages such as Go and Rust have Haiku support, but are in the form of patches and Haiku still misses out on being a tier 1 target for these languages. Fortunately, Swift 3.1's focus on portability makes it possible to add Haiku as a target just like other platforms such as Linux, FreeBSD and Android due to mostly being POSIX compliant. Therefore, I believe that this proposal to port Swift 3.1 to Haiku and to prepare these patches for merging upstream is a feasible project to complete within the GSoC period.

About Swift

According to the Swift.org website, it defines Swift as *'a general-purpose programming language built using a modern approach to safety, performance, and software design patterns.'*(1) Swift has several features found in other languages such as closures, generics, tuples, type inference, etc but includes exclusive features such as optional types that checks a variable that *"there is a value and it equals x or there isn't a value at all"* (2), which always initialises values before using them as attempting to directly set a object to a null pointer is disallowed in Swift.

Since Swift 2.2 was open-sourced in December 2015, it has been ported to a number of platforms such as Linux, Android, FreeBSD and Cygwin. Moreover, the growing ecosystem and popularity of the language is hard to ignore, such that it is already among one of the most popular languages according to the 2016 StackOverflow Developer survey (3) and was recently ranked 10th in the TIOBE index (4). Swift can be used for various purposes ranging from teaching programming to systems programming or server-side development.

Goals

The bare minimum components that need to be ported in-order to make Swift functional on Haiku (excluding external dependencies and the forked versions of LLVM

and Clang)(5), are:

- The swift compiler (swiftc)
- The standard library (stdlib)
- Swift Core Libraries framework(s) (swift-corelibs-foundation)

After reviewing the required fixes to both Haiku and Swift, the realistic goals that can be met within the duration of GSoC are as follows:

- Building the compiler and standard libraries by using swift-clang and swift-llvm.
- Porting the core libraries to Haiku.
- Ability to compile / run swift programs built with swiftc.
- Testing the built compiler and add Haiku as a stdlib test (stdlibTest) platform.
- Clean up platform support and prepare changes for upstreaming.

For the best chances of fulfilling this proposal, I have already patched (swift-llvm, and swift-clang) so that they can build themselves, and patched the 'build-script'(1A) and added a early recipe (1B) for Haiku support which will be available on my GitHub forks (6). The FreeBSD port based on Swift 2.2 (7)(8) will be used as a reference for adapting the specific libraries that Haiku needs.

Project Timeline

My suggested plan for achieving these goals are heavily based on rough estimations. Because of this, I set a maximum limit on the weeks spent on a subgoal.

Community Bonding Period – (May 1st - May 30th)

During the community bonding period, I'll introduce myself to both the Haiku and Swift communities on the project whilst familiarising with the swift compiler architecture and to discuss with my mentors on building swiftc. I'll allocate 6 - 7 hours every day in patching and communicating with my mentors.

Building requires the forked LLVM/Clang components, as they have swift-specific compiler attributes added:
swift_private, swift_name, swift_error and swift_newtype, which are all used to bridge between Objective-C and C declarations into Swift. (10)

For upstream support, I intend to support only x86_64 for Swift. While it is possible to port to 32 bit systems in the past, (11) (in the case for Linux) if this doesn't interfere with Swift.orgs' goals, then supporting 32 bit may be considered in the future.

I intend not to take time off during this period, The weekly schedule for this timeline are as follows:

Week 1/2 (May 4th - May 18th)

- Add Haiku x86_64 platform recognition in build_script
- Add Haiku as a standard library target
- Add Haiku support for validation tests
- Complete full platform build-script support for Haiku.

File(s) related to this task: **build-script.py and build-script-impl.py toolchain.py, targets.py, AddSwift.cmake, FindICU.cmake, CMakeLists.txt**

I'll discuss with my mentor(s) on the recommended changes for integrating a new platform and will allocate a maximum of 2 weeks for build-script support.

If this is achieved in less than that, I'll add more platform support for the standard libraries, runtime and to add stdlib-validation-tests for Haiku (In validation-test/StdlibUnittest).

Week 3 (May 19th - May 25th)

- Investigate corelibs-foundation library support (Foundation)
- Discuss libdispatch support for Haiku (optional).

I'll be attempting to build the swift compiler with these changes. I'll expect some errors in the compilation process, possibly in the lib folder which contains the LLVM compiler backed that swift uses. I'll ask my mentor(s) about the specific areas that need patching for upstreaming, before porting it.

Week 4 (May 26th - June 1st)

- Support optional build-script features
- Submit initial Haiku support patches upstream.

- Update Swift recipe.

LLVM/Clang based patches may only be for haikuports to compile the swift toolchain.

1st Coding Period – (June 1st - June 30th)

The Swift compiler backend in the /lib folder contains the Parser, AST and the SIL (Swift Intermediate Language) and the rest of the compiler internals which is written in C++. After porting the compiler backend, swiftCore and stdlib will later be ported. They contain the standard objects available in all swift programs such as 'String, Array, and Collection'. The standard library core found in 'stdlib/public/core' is written in Swift itself (12), but fortunately the standard library core does not contain any platform-specific code present in the core so porting it shouldn't be a problem. I estimate a maximum of two weeks to port both of them to Haiku.

After June 16th, I can dedicate more time (60 Hours pw) working in this period, as I'll be free for more than a month.

Week 5/6 (June 2nd - June 16th)

- Modify the compiler backend (lib/Driver and lib/Basic) for Haiku support.
- Build swift toolchain using Clang and LLVM.
- Run tests against swiftc via the integrated testsuite.
- swift test-cases for Haiku.

File(s) related to this task:
- Swift/lib/Driver/ToolChains.h
- Swift/lib/Basic/Unix/TaskQueue.inc
- stdlib/public/Platform/Platform.swift
- stdlib/public/Core/CTypes.swift
- stdlib/private/SwiftPrivatePthread
- stdlib/private/StdlibUnitTest

With each of these changes made, they will be tested and refactored for correctness which will be reviewed by my mentor on Haiku's POSIX layer and also reviewed by the swift maintainers so that the new compiler, runtime and it's associate libraries are tested to work as expected.

Week 7 (June 17th - June 23rd)

- Continue to test swift against testsuite (unittests)

Week 8 (June 24th - July 1st)

- Review compiler patches with mentor(s)
- Refactor/Finalise patches and merge swift compiler support upstream.
- Update recipe with the aforementioned changes.
- Focus on bringing up Foundation & CoreFoundation support.

I'll discussing the platform requirements with my mentor(s) on porting the core-libs before July.

2nd Coding Period – (July 1st - July 30th)

The 'Foundation' Framework in the Swift core libraries (13) provides basic utility classes outside of the runtime and porting this will be the most challenging part of this period. Written in C, it servers as a re-implementation of Apple's Foundation framework found in macOS and iOS platforms but independent of using the Objective-C runtime; thus using POSIX libaries instead to increase portability. Although it isn't required to build swift, it will be required if a developer wishes to use an external library that depends on it.

During July 1st to July 16th I'll allocate 10 hours a day in focusing to port CoreFoundation, since I'll be still off from work. Therefore, I can dedicate into spending my time porting these libraries.

Week 9 (July 2nd - July 9th)

- Add the Haiku platform SDK to build.py and target.py
- Add platform definitions to CoreFoundation library files.

File(s) related to this task:
- build.py
- target.py (Platform Target)
- CoreFoundation/* (Lower level interface to Foundation.)

Week 10 (July 10th - July 18th)

- Add Haiku as a platform in Host.swift and Platform.swift
- Patch the higher level Foundation libraries.
- Compile applications using foundation with swiftc
- Build Foundation and test functionality via swift-corelibs-xctest

File(s) related to this task:
- Host.swfit, Platform.swift (Build support)
- target.py (Platform Target)
- CoreFoundation/* (Lower level interface to Foundation.)

Week 11 (July 19th - July 26th)

- Add networking support via NSURLSession libraries.
- Implement/Add missing library support (NSDate/Process)

File(s) related to this task:
- Processes.swift (Part of process spawning test cases)
- Subprocess.swift (Supported in standard library)

Process spawning related modules such as TestProcess.swift, Processes.swift and Subprocess.swift all use the standard 'posix_spawn()' function which are inculded from spawn.h libraries and are currently unimplemented in libroot. Possible workarounds for this are:

- Disable support for spawning for now.
- Use fork+exec() for spawning processes.
- Implement spawn.h library for Haiku. (Preferred)

Implementing spawn.h was mentioned in the chat logs earlier (14). I'll discuss this with my mentor on doing this properly (Using BRoster::Launch / runtime_loader functions) so that it can pass the sub-processing library tests.

Week 12 (July 27th - August 3rd)

- Clean up and prepare CoreFoundation patches for review.
- Update swift recipe to reflect more changes.

3rd Coding Period (August 1st - August 31st)

Week 13/14/15 (August 4th - August 18th)

In August, I plan to further test swiftc by using the build-script and to add Haiku as a test platform for the standard library. These tests will cover POSIX functionality (15), where I expect issues when testing on Haiku.

Hence this, I'll be using Haiku's open-posix test-suite that covers POSIX compliance on standard functions that are defined in the operating system. I will aim to test the CoreFoundation / Libraries for correctness as well as the standard library under the tests that swift provide for about 2 - 3 weeks, as these tests will be needed for correctness of all components. If required, I'll add the test cases for the Haiku platform.

Week 16/17 (August 19th - August 31st)

I'll spend 2 weeks on cleaning up some platform code, fixing bugs encountered during the porting process and finalising my patches for merging with the stable branch. From there, the swift recipe should be able to build and compile the swift toolchain without fail within the duration of this project.

After Google Summer of Code

This porting task deals with the essential components to get Swift functional on Haiku and ready to be a supported target. There is still more to do after GSoC (LLDB for Swift REPL support, libdispatch for threading, etc. Even after GSoC I will continue to improve Haiku support for Swift and will certainly continue to contribute to Haiku by both sending patches via Trac, and porting more sophisticated software. Furthermore, once Swift support for Haiku is merged, I seek to further study Haiku's internals to help in supporting newer architectures.

About

I'm a second year university student studying Computer Science at the University of Hull with interests in OS internals, compilers, embedded devices, computer security and contributing open source/FLOSS projects. In my spare time I contribute via the haikuports organisation to support more third party applications and libraries

for Haiku.

I have experience in developing software in C, C++, Objective-C, Go and Python on multiple platforms. Since late 2014, I've used Haiku on a USB for mainly learning the Haiku API for writing native Haiku apps.

Why Haiku?

One of the reasons I selected the Haiku project was mainly due to it's core philosophy for being an alternative desktop OS for all-types of users whilst promoting usability, performance and retaining BeOS R5 app compatibility. But the one idea that distinguishes it from other open source OSes is its software components are tightly integrated with the OS from the bootloader to the applications, which is considered rare for a open source OS project. Due to this level of consistency I have been using it ever since.

On the developer perspective, I have looked at the Haiku Book and BeOS: Porting UNIX applications book to join in the porting efforts. There was some BeOS Sample code from Be Inc, which was useful for testing out BeOS APIs but trying some of them with Haiku obviously failed to compile. So I am currently hosting the sample code for archiving purposes and for everyone to see on Github, with the aim of porting it from BeOS to Haiku for my own learning and to try on Haiku.

Expectations from Mentors

As this proposal involves knowledge around libroot, runtime_loader and the posix layer, I would expect some expertise from my selected mentors around those particular areas. This project might involve modifying the posix layer to solve some of the tests that is built into swift.

References

(1) Swift.org, (2015), [online] Available at: https://swift.org/about/
[Accessed 23 Mar. 2017].

(2) The Swift Programming Language, (2017), 3rd ed, [ebook] Cupertino, CA: Apple Inc, 35, Available at: https://swift.org/documentation/ [Accessed 23 Mar. 2017].

(3) Stack Overflow, (2017), Stack Overflow Developer Survey 2017, [online] Available at: https://stackoverflow.com/insights/survey/2017#most-loved-dreaded-and-wanted [Accessed 22 Mar. 2017].

(4) Tiobe.com, (2017), TIOBE Index — TIOBE - The Software Quality Company. [online] Available at: https://www.tiobe.com/tiobe-index/ [Accessed 22 Mar. 2017].

(5) https://swift.org/source-code/#cloned-repositories

(6) https://github.com/return/swift-llvm/tree/swift-3.1-haiku

(7) https://lists.swift.org/pipermail/swift-dev/Week-of-Mon-20151207/000294.html

(8) https://github.com/apple/swift/pull/4804

(9) https://github.com/apple/swift/tree/swift-3.1-RELEASE/cmake/modules

(10) http://lists.llvm.org/pipermail/cfe-dev/2015-December/046335.html

(11) https://lists.swift.org/pipermail/swift-dev/2015-December/000030.html

(12) https://github.com/apple/swift/tree/swift-3.1-RELEASE/stdlib/public/core

(13) https://swift.org/core-libraries/#foundation

(14) https://echelog.com/logs/browse/haiku/1377986400

(15) https://github.com/apple/swift-corelibs-foundation/blob/master/TestFoundation/TestProcess.swift

(1A) https://github.com/return/swift/commit/c8253ab3afa238c16ca9389d169c0659dc10b520

(1B) https://github.com/return/haikuports/commit/66531b0698259deeca001c73e4e3f87f9dcf9115