Introduction

I am Trung Nguyen, also known as @trungnt2910 on GitHub and other social media sites. This year, I am a first-year computer science student at the University of Wollongong in Australia.

I first discovered Haiku as a participant of the Google Code-in 2019 event. At that time, I was an absolute beginner in C/C++ and I wanted to have a chance to see what developing an operating system and its associated userland application looks like. Therefore, I chose to complete tasks in Haiku.

As I use Haiku, I see that this operating system has a rich ecosystem of applications while retaining its simplicity and consistency in design. At the lower levels, Haiku has an API that is standard-conforming enough to make porting open-source software straightforward, but also unique enough to make this activity fun.

Following the Google Code-in journey with Haiku, I've been continuing my active involvement in the open-source community. I've made code contributions to several C# UI frameworks (the Uno Platform, GtkSharp, and Microsoft's own .NET MAUI). I've worked a lot with C/C++ code through emulator projects such as Darling (MacOS userspace emulator), blink (Linux x86_64 binary emulator), and Hyclone (my own Haiku userspace emulator). I've also made contributions to the Haiku kernel, libroot, and written a few recipes for HaikuPorts.

Project proposal – .NET Developer Platform port for Haiku

Background

.NET (dotnet) is an open-source cross-platform runtime. It is popular among developers: According to StackOverflow 2021 Developer Survey, .NET is the most loved framework. C#, the primary language used with .NET, is also a popular language, constantly being in the TIOBE index's top 10.

There has also been requests for support C#, Mono and .NET among Haiku users, such as this forum thread, or this forum thread.

Porting .NET to Haiku would not only provide users with a way for users to run more apps on Haiku, but also opens up an entirely new method of creating native Haiku apps: Using the Haiku API through C# wrappers and developing either directly on Haiku with the dotnet CLI, or cross-compiling from an IDE on one's favorite operating system.

Current state

Currently, some progress has been made with a Haiku port of .NET. Essential dependencies has been ported to Haiku and available on HaikuPorts, and the CoreCLR is capable of running a .NET 7.0 "Hello World" binary. However, as of March 2023, the build has been broken and does not work on Haiku r1-beta4.

Dependencies

- Dependencies for building .NET 7 on Haiku are already available on HaikuPorts. The most notable work is porting krb5 and llvm-libunwind.

Compilation

- A cross-compilation environment has been created for Haiku and officially merged to the official .NET repo. It is available here. However, the build image is pinned at a specific commit and hrev, which should be updated to r1-beta4.
- Currently, compiling .NET requires a working version of .NET installed on the build machine in order to generate managed .NET `.dll` files. Therefore, .NET for Haiku cannot be built directly on Haiku yet and requires a Linux machine.

CoreCLR and System libraries

- The PAL (Platform Abstraction Layer) has been ported to Haiku. The artifacts have successfully been compiled on a Ubuntu 22.04 Linux machine. The tests for this layer worked on Haiku at that time when copied to a Haiku machine running a pre-beta4 nightly build. However, as .NET is a rapidly evolving platform, it is not certain that it still works for .NET 8 without any further modifications.
- Some system libraries are also working, enough for the .NET API `System.Console.WriteLine` to function on the Haiku port.
- A C# “Hello World” binary could be run on Haiku if some appropriate artifacts are copied to the same Haiku machine (for example, through `scp`). However, due to changes to the JIT system, as of March 2023, it no longer works.
- None of these patches have been upstreamed to the official .NET repository.

Scenarios – What the end user should be able to do after this project

Installing .NET on Haiku

Users can easily download and install .NET through the familiar pkgman command:

```
pkgman install dotnet
```

Native .NET development

The dotnet CLI can support development natively on Haiku. This means command operations like `dotnet restore`, `dotnet run`, `dotnet build` should work just like on any other OSes.

Cross-compilation

Developers with existing applications can simply copy their `net8.0` console applications to Haiku and expect them to run if the application only uses .NET base class libraries and does not call into any native or system-specific APIs.

Developers who want to build apps for Haiku or make a Haiku-specific version of their Haiku applications can create a `net8.0-haiku` C# project by installing a Haiku-specific workload. This “workload” contains definitions of the Haiku API, similar to C++ header files. However, .NET workloads do not require a dedicated cross-compiler; developers can use the same toolchain on their host machine.

net8.0-haiku
A .NET 8.0 Haiku workload and its corresponding TFM (Target Framework Moniker), `net8.0-haiku`, can be referenced by users. Like other workloads, users can install it through NuGet, through a setup script provided in a repository in HaikuArchives, or through pkgman.

The .NET Haiku workload gives users with a set of C# language bindings for the Haiku API, such as those provided by `libbe.so`. For example, a minimalistic Haiku application can be created like this:

```
using Haiku; // Most C# libraries/frameworks keep their classes in a separate namespace

var a = new BApplication("application/x-vnd.your-app-sig");
a.Run();


// a's lifetime would be handled by the garbage collector.
```

The accompanying `.csproj` file would look like:

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>net8.0-haiku</TargetFramework>
  </PropertyGroup>

</Project>
```

Users using `net8.0-haiku` can also enjoy the rich variety of .NET libraries available in the NuGet ecosystem as `net8.0-haiku` is compatible with `net8.0` and any other TFMs compatible with `net8.0`.

Requirements – Project technical goals/non-goals

Goals

- Rebase existing work on top of the next .NET version (.NET 8). Port any new dependencies to Haiku and ensure that the PAL tests still pass.
- Porting the .NET libraries and CLI tools.
- Ensure that .NET behaves as expected on Haiku through the included test suite. Fix any Haiku kernel/libroot bug that prevents this from happening.
- Generating .NET bindings for the Haiku API.
- Port some popular .NET-based frameworks. GtkSharp, FNA, and ASP.NET core are popular and potentially useful targets.
- Create HaikuPorts recipes for .NET, its dependencies, and some of its major frameworks/workloads.

Non-goals

- Porting a .NET debugger. The commonly used debugger, vsdbg, is not open source.

- Porting frameworks that require too much system-specific code. For example, .NET MAUI on top of Haiku API is possible but out-of-scope for this proposal.

Technical overview

Related Git repositories

- https://github.com/dotnet/runtime: Official .NET Runtime repository. This contains all core components of .NET: CoreCLR, libraries, and the .NET CLI tool.
- https://github.com/dotnet/arcade: This repository contains some build scripts shared among various .NET repositories. A script to set up a .NET build environment for Haiku should be here.
- https://github.com/trungnt2910/dotnet-runtime: A partial work on porting .NET in 2022.

When porting additional frameworks, other repositories might also be involved. For example, ASP.NET Core is located at https://github.com/dotnet/aspnetcore.

Build environment

- Building .NET requires .NET installed on the build machine. A large part of .NET's build script is based on MSBuild. Furthermore, .NET system libraries are written in C#. Therefore, a platform already supported by Linux is required. For this purpose, Ubuntu Linux 22.04 (on WSL1) is used.
- A copy of Haiku cross compilation tools and a Haiku rootfs is required. This can be obtained either using .NET's official build-rootfs.sh script, or using existing installations shared with other activities (such as the cross-tools used to build Haiku and the rootfs generated by a HyClone prefix).

Programming languages involved

Most code involved is written in C or C++. Occasionally some C# is used, mainly to implement Haiku platform-specific code that works under existing abstractions.

Assembly *may* be involved, but as Haiku has a normal Sys-V ABI existing code for other UNIX systems should work.

Expected problems

- .NET is an ever-evolving platform. It is expected that after a year, some optimizations that .NET has might break the existing Haiku changes.
- The Haiku toolchain has an exception handling bug when cross-compiled from Linux. A workaround is to explicitly link `-lgcc_s`.
- A certain new feature introduced in .NET 7, *GC regions*, tries to reserve 256GB of virtual memory. The `MAP_NORESERVE` flag should be added to `mmap` when such things happen on Haiku.

Step by step reproduction

(Based on instructions by @jessicah)

On the Linux build machine:

```
export BUILDROOT=$HOME/builds/cross-compiler
```

```
git clone https://github.com/jessicah/cross-compiler
cd cross-compiler && ./build-rootfs.sh x86_64 --rootfsdir $BUILDROOT
for file in *.hpkg ; do $BUILDROOT/package_extract.sh $file ; done
git clone https://github.com/jessicah/dotnet-runtime
cd dotnet-runtime
# make the Haiku cross-compiler available
export PATH=$PATH:$BUILDROOT/generated/cross-tools-x86_64/bin
# make the Haiku sysroot available; libraries and headers will be in here
export ROOTFS_DIR=$BUILDROOT
git checkout haiku-dotnet7
./build.sh clr.runtime -arch x64 -os haiku -c debug -cross
./build.sh mono.runtime -arch x64 -os haiku -c debug -cross
./build.sh libs.native -arch x64 -os haiku -c debug -cross
./build.sh host.native -arch x64 -os haiku -c debug -cross
./build.sh clr.paltests -arch x64 -os haiku -c debug -cross
```

Then, on a Haiku machine:

```
cd dotnet-runtime
src/coreclr/pal/tests/palsuite/runpaltests.sh artifacts/bin/coreclr/$(uname).x64.Debug/paltests
```

Timeline

Community bonding period (May/June):

- Rebase prior work at https://github.com/trungnt2910/dotnet-runtime to the latest development branch of .NET.
- Port new dependencies (if any).
- Fix any bugs in the PAL and CoreCLR, or libroot and Haiku if it is a bug on the Haiku side. Try to reproduce the "Hello World" binary example on the latest .NET 8 preview and on the latest Haiku.
- Start opening pull requests to .NET for:
    - Configurations (*.sh, *.props, *.targets, *.*proj, .cmake, eng/native/ etc.)
    - CoreCLR (src/coreclr)
    - Mono (src/mono)

First month of coding (June/July):

- Port the .NET host (src/native/corehost & src/installer) and open a pull request for that.
- Port the runtime libraries and tests (src/libraries) and (src/tests). Open a pull request when this is completed.
- Fix any unexpected bugs on Haiku's side (libroot, kernel).
- Satisfy any requirements from .NET reviewers.
- Obtain a binary copy of .NET, and work on building it natively on Haiku.

Second month of coding (July/August):

- Satisfy any requirements from .NET reviewers.
- Fix any broken tests.
- Finalize the native .NET build on Haiku.
- Start developing the `net8.0-haiku` workload.

Third month of coding (August/September):

- Create instructions on how to use/install the `net8.0-haiku` workload. Receive community feedback and fix any bugs for the workload and .NET Haiku API bindings.
- Port and test common .NET-based frameworks: GtkSharp, ASP.NET Core,...
- Satisfy any requirements from .NET reviewers.

After Google Summer of Code:
In the most optimistic case where this project gets merged to .NET before .NET 8, it will have to wait until November for an official release. Then a .NET recipe can be created on HaikuPorts.

Expectations from mentors

.NET is a piece of software that requires a lot of specific behavior from its host operating system. Therefore, I expect some help from mentors in navigating some deeper Haiku internals.

Furthermore, as the project involves a lot of cross-compiled code, I would appreciate instructions on how to efficiently move binaries between machines, as well as improving the debugging experience (as debug symbol resolution does not work properly without some configuration).

Miscellaneous

It might not seem credible to claim to be able to work 56 hours/week. However, in reality, I do spend that much time on open-source projects even without getting paid. The first semester of university does not have a challenging workload, some of the tasks for subjects like *Fundamental Programming with Python* can be quickly completed by someone with prior programming experience.

My activity on GitHub as well as my activity on Haiku during summer 2022 should prove that I can fit a 350-hour project in my schedule.