Introduction

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

I have made contributions to various areas of Haiku development since 2019, as part of Google Code-in 2019, Google Summer of Code 2023, and occasionally during my free time as well.

For my last GSoC project, I want to work on a problem encountered throughout my whole journey with Haiku: Userland debugging issues.

Project proposal – Improving the userland debugging experience

Background

Currently, the native `Debugger` app is the only known working tool for local userland debugging on Haiku. However, this tool has various bugs and limitations, hindering the process of building complex applications, especially complex ports.

Moreover, for many developers working with a UNIX-like environment, the most familiar debugger would be either GDB or LLDB. These two programs also come with lots of features that are not present in `Debugger`, such as Python scripting.

Providing an alternative debugger choice would also speed up the process of fixing `Debugger` bugs, especially cases when `Debugger` crashes when trying to debug itself.

Current state

An ancient version of GDB (6.3) has already been ported to Haiku. This version is known not to work with the latest Haiku x86_64 images. A few years ago, a recipe for GDB 8.1 has been uploaded to HaikuPorts but is not available to the public and contains quite a few bugs preventing the debugger from starting processes.

Separate efforts have also been made to port LLDB 12, but the port is not known to work either.

Requirements – Project technical goals

- Have the latest GDB (currently 14.1) running with all applicable local debugging features working on x86_64 Haiku.
- Have the latest LLDB corresponding to the latest version of LLVM ported to Haiku (currently 17) running with all applicable local debugging features working on x86_64 Haiku.
- Improve the general stability of Haiku's `Debugger`, such as letting it debug large binaries such as CoreCLR without crashing and lagging.
- Improve `Debugger`'s ability to handle symbols. Allow browsing for symbol files during debugging and browsing for source file paths to serve cross-compiled binaries.

Technical overview

Related Git repositories

- https://sourceware.org/git/binutils-gdb.git: Official GDB repository.
- https://github.com/llvm/llvm-project: The official LLVM monorepo, containing LLDB.

- https://github.com/kenz-gelsoft/llvm-project/tree/haiku12: A work-in-progress port of LLDB that is not working yet, may serve as guidance for LLDB's project structure.
- https://github.com/haikuports/haikuports: HaikuPorts repos, contains previous porting attempts and also stores future successful patchsets.

Programming languages involved
Most code involved is written in C++. Some assembly may be involved to handle low-level architecture-specific parts.

GDB/LLDB interface overview
General Haiku issues
- Haiku uses Sys-V ABI, POSIX API, ELF binary format, and DWARF debugging symbols, like many other UNIX targets.
- Haiku does not have any kind of `ptrace`-style debugging API. Haiku's debugger API is unique:
    - All debugging commands, results, and events are communicated through a pair of ports.
    - Reading the debug port is nearly equivalent to a `ptrace`/`waitpid` loop, or `WaitForDebugEventEx` on Windows.
    - Both debugger supports Windows, a completely "exotic" operating system compared to the *NIXes, so they should have adequate abstractions to handle most Haiku differences.
    - Some special handling will be needed for debug operations since unlike ptrace, results are returned asynchronously and potentially mixed with debug events.
- Despite mostly following the POSIX API, potential bugs/differences in the kernel or libroot have been known to cause issues in many software ports, especially in parts related to virtual memory and process lifetime.

GDB
Most Haiku-specific code should go in a few files:

- `haiku-nat.c`: "Native-dependent" code. Handles process/thread lifetimes and debugging functions. Code here should override the `target_ops` class.
- `haiku-tdep.c`: "Target-dependent" code. Handles low-level ABI-specific aspects, such as registers, frame unwinding, and signals.

Compared to the patch in HaikuPorts, GDB's design has significantly changed, most noticeably by using C++ classes and inheritance.

LLDB
The LLDB source tree contains:

- `Host/**`: Contains abstracted debugging logic and general system functions (creating pipes, locks, gathering system information) to run on the host machine. Haiku should be able to share most of POSIX code.
- `Plugins/Platform`: Contains support functions for the target machine, including spawning processes, loading libraries, and fetching certain native constants.

- `Plugins/Process`: Contains the actual debugging logic: Spawning processes, setting breakpoints, stepping, modifying memory. This is where most Haiku-specific code should be.

Debugger improvement overview
The Haiku Debugger consists of a few parts:

- `src/apps/Debugger`: The front-end for Haiku's debugger, including a GUI and a CLI. Work on UI issues or improving the settings experience should be done here.
- `src/kits/Debugger`: The "Debugger Kit". Implements most debugging operations, such as parsing symbol files, disassembling, walking the stack frame, and so on. Most improvements to `Debugger`'s capabilities should involve work in this part.
- `src/kits/Debug`: The "Debug Kit". This is mostly a wrapper of the low-level API to C++ classes.

Some reported `Debugger` tickets that also affect my prior software porting work include:

- #8877: *Stacktraces without debuginfo*. A possible solution is to compare the `x86_64` implementation in the Debugger Kit with the `x86` counterpart and implement the missing parts.
- #11596: *No disassembly for functions written in assembly*. Some work might be needed to decouple `Debugger`'s UI and the Debugger Kit disassembly component from symbols, just like how GDB and even WinDbg allows the user to view arbitrary instructions before or after the program counter, annotating with symbols when appropriate.
- #15087: *Permanently skip request to install 3rd party debug information*. This should be a simple front-end fix, in addition to the existing option to skip debug information for the current session.

Timeline

Community bonding period (May/June):

- Examine the existing GDB patchsets.
- Try to build and fix any obvious bugs with the GDB 8.1 port.
- Start with a fresh GDB 14.1 port.
- Test debugging with GDB 14.1:
    - Process creation/termination
    - Symbol loading and resolution
    - Stack trace
    - Disassembly
    - Scripting

First month of coding (June/July):

- Complete and submit GDB recipe to HaikuPorts.
- Fix any unexpected bugs with GDB.
- Improve `Debugger`'s stability using the newly ported GDB.
- Improve `Debugger`'s symbol resolution.
- Find `Debugger`-related Trac tickets or probe for community ideas.

Second month of coding (July/August):

- Fix as many `Debugger`-related issues as possible.
- While waiting for code review, start a fresh port of LLDB, based on LLVM 17.x, or a newer version of LLVM if upstreamed to HaikuPorts by that time.

Third month of coding (August/September):

- Fix remaining `Debugger` issues.
- If there is still remaining time, complete the LLDB port and test debugging with LLDB.
- Upstream Haiku-specific LLDB changes to the LLVM Project.

After Google Summer of Code:
Continue collaborating with reviewers from relevant repositories to upstream any remaining code.

Expectations from mentors

Since the idea of porting GDB and LLDB to Haiku is not new, yet previous attempts have mostly failed, some insights from mentors about mistakes and other experience from older ports would greatly assist this project.