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.

I have a passion for trying out unique platform/software combinations to optimize for specific developer or user scenarios, either by writing compatibility layers or improving the components' codebase to allow them to work together. Recently, my interest is focused on software working with the ARM64 architecture.

Project proposal – ARM64 Port for Haiku

Background

In recent years, ARM64 has been a trending competitor to the more traditional Intel platform. Its success is most evident in the mobile devices and embedded systems market, while slowly taking on larger form factors such as laptops. Nowadays, all major operating systems currently have ARM64 support, as well as popular software development tools.

An ARM64 port has also been in progress for Haiku since 2018 and has attracted significant interest from the community.

When working properly on ARM64, Haiku could be deployed on a much wider range of devices with the help of EDK II EFI implementations, including system-on-chip boards and mobile phones, given the appropriate drivers. Haiku virtual machines could also be more efficient on these devices using virtualization frameworks such as Hyper-V, KVM, or HVF. Furthermore, an ARM64 Haiku would also allow running, testing, and porting more HaikuPorts recipes, diversifying Haiku's app ecosystem.

Current state

Progress has been made to port Haiku to ARM64 since 2018. The build system has been configured, allowing the compilation of minimal Haiku images. The kernel can be booted on a `qemu-system-aarch64` virtual machine with EDK II, nearly reaching `runtime_loader` and the userland. However, the port is nowhere near a usable state, with stubs scattered across the kernel and userland.

Kernel

- When transitioning to userland, the kernel panics with "`arch_vm_translation_map_create_map user not implemented`". It seems like the virtual memory mapping is not properly implemented yet.

Userland

- With a compatibility layer like HyClone, the Haiku ARM64 userland can be tested early without a working kernel.

- Every binary compiled for Haiku fails to load due to misconfigurations in `runtime_loader` and more importantly, a lack of support for ELF relocations, making the program crash during binary initialization.
- Stubs are also found in `libroot`, such as for `system_time()`.

Requirements – Project technical goals

- Kernel stably boots and runs on `qemu-system-aarch64`, both emulated from `x86_64` and natively through KVM.
- Userland applications, including the application server, system information tools like `sysinfo`, and development tools like `strace` and `debugger`, function properly, comparable to more mature ports such as `riscv64` and `x86_64`.
- Users should be able to compile/download Haiku, set up a virtual machine, and download essential packages. These packages should work reasonably compared to other architectures.

Technical overview

Related Git Repositories

- https://git.haiku-os.org/haiku: The main Haiku repository.
- https://github.com/haikuports/haikuports: Required for building, updating, and fixing ARM64 ports.

Build environment

For this project, I will be using Ubuntu 23.04 on `x86_64` WSL1 to cross-compile Haiku. I will also sometimes use Ubuntu 22.04 on my `aarch64` Orange Pi 5 Plus board when compiling images for it.

Theoretically, any environment that can be used to build `x86_64` Haiku should be supported.

Programming languages involved

Most code involved is written in C++. Some ARM64 assembly may be involved to handle low-level architecture-specific parts.

Kernel - ARM64 Memory Management

An overview documentation for ARM64 memory management can be found here.

Memory management on ARM64 follows the general strategy seen in most processors: A table mapping addresses of virtual pages (usually 4KB, but can be configured to be 16KB or 64KB), to physical pages. This is consistent with Haiku's architecture-independant logic of mapping `addr_t` of areas to `phys_addr_t`. While ARM64 does have the concept of multiple address translation tables for different privilege levels, the additional tables are only used for hypervisors and are transparent to the guest OS. For now, Haiku only needs to handle the tables at `TTBR0_EL1` and `TTBR1_EL1`.

All architecture-specific code for ARM64 memory management on Haiku lies in `src/system/kernel/arch/arm64`. `arch_vm.cpp` contains initialization hooks and functions returning mapping capabilities and characteristics. `arch_vm_translation_map.cpp` contains the interface for VM translation mapping, which then calls into C++ code (`PMAPPhysicalPageMapper.cpp` to obtain virtual addresses, and `VMSAv8TranslationMap.cpp` to do the actual mapping).

Orange Pi 5 Plus - Driver support status
Based on the device tree for Orange Pi 5 Plus, some essential devices are, to some extent, supported on Haiku:

- UART: Required for debugging in the early stages. uart0 is the generic `"snps,dw-apb-uart"` device, which is supported on Haiku.
- USB: Required as a storage medium during installation and for usage when other methods are not supported yet, as well as to connect peripherals like keyboards and mice. The Orange Pi 5 Plus implements USB 1.0 (OHCI) and USB 2.0 (EHCI), both of which are standards implemented on Haiku. It also supports dual-role USB 3 (DWC). While DWC is not supported on Haiku, it is compatible with XHCI for host mode, making it usable on the OS.
- Ethernet: Required for establishing internet connection. Orange Pi 5 Plus uses the RTL8125 adapter, which has already been ported to Haiku.

Other OSes, such as NetBSD, are already running on RK3588 chips in ACPI mode on EDK II, giving us a potential reference implementation.

Step-by-step reproduction
This is mostly based on the official build guides.

From an Ubuntu environment:

```
# Install the prerequisites:
sudo apt install -y git nasm bc autoconf automake texinfo flex bison gawk build-essential unzip wget zip less zlib1g-dev libzstd-dev xorriso
libtool python3

# Build the cross tools
cd haiku
mkdir generated.arm64; cd generated.arm64
../configure -j$(nproc) --cross-tools-source ../../buildtools --build-cross-tools arm64

# Build the raw disk images for EFI booting
jam -j$(nproc) -q @minimum-raw esp.image haiku-minimum.image
```

After building, `esp.image` should be at `generated.arm64/objects/haiku/arm64/release/efi/system/boot/esp.image`, while `haiku-minimum.image` is placed right at `generated.arm64`.

The built image can then be run on QEMU:

```
qemu-system-aarch64 -bios /usr/share/qemu-efi-aarch64/QEMU_EFI.fd \
    -M virt -cpu max -m 2048 \
    --accel tcg,thread=multi \
    -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 \
    -device virtio-blk-device,drive=x1,bus=virtio-mmio-bus.1 \
    -drive file=/path/to/haiku/generated.arm64/objects/haiku/arm64/release/efi/system/boot/esp.image,if=none,format=raw,id=x0 \
    -drive file=/path/to/haiku/generated.arm64/haiku-minimum.image,if=none,format=raw,id=x1 \
    -device virtio-keyboard-device,bus=virtio-mmio-bus.2 \
    -device virtio-tablet-device,bus=virtio-mmio-bus.3 \
    -device ramfb -serial stdio
```

For ARM64 devices, replace `-cpu max` with `-cpu host`, and `--accel tcg,thread=multi` with `--accel kvm`.

Timeline

Community bonding period (May/June):

- Study about ARM64 memory management using documentation and code samples from other operating systems such as FreeBSD.
- Study, test, and finalize the existing work-in-progress code for ARM64 MMU at #7523.

- Fix any obvious bugs and implement easy stubs.

First month of coding (June/July):
- Boot the Haiku minimum image at least until `app_server` launches on QEMU.
- Generate packages needed to build the Haiku full image.
- Create regular builds of ARM64 Haiku for community testing and work with Haiku infrastructure developers to enable ARM64 nightly images.

Second month of coding (July/August):
- Ensure the proper functionality of essential userland applications (`app_server`, system information monitoring, `Debugger`).
- Test Haiku on an EFI-enabled Orange Pi 5 Plus (RK-3588 chip) using U-Boot.
- Write or port relevant drivers for this board.
- Fix any remaining kernel issues.

Third month of coding (August/September):
- Port and build popular HaikuPorts packages for ARM64, then find a way to deliver them to `pkgman`.
- Fix any remaining issues with kernel and system applications/libraries.

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

Expectations from mentors
While not new to the Haiku codebase, I do not have much experience with code directly touching hardware such as the MMU. I would highly appreciate quick pointers to relevant documentation and code samples from other architectures or other OSes.