FreeBSD compatibility layer proposal for GSoC 2026

Personal info:

Full name: Pranav Bisht
Timezone: IST (GMT+5:30)
Email address: Pranav0bisht@gmail.com
IRC username: iamdumb
Trac username: iamdumb
Gerrit changes submitted:

1. fixed comparison warnings(https://review.haiku-os.org/c/haiku/+/10363)

Will you treat *Google Summer of Code* as full time employment?
Ans: Yes.
How many hours per week will you work?
Ans: 50 hours per week.
List all obligations that may take time away from GSoC.
Ans: Semester end examination from May 22, 2026, to June 18, 2026. So my time allotted to gsoc will be cut in half during this time.
Are you using *Google Summer of Code* to fulfill a university requirement -- internship, class credit, ..?
Ans: No.
How and when did you first heard about Haiku?
Ans: i first came across haiku in january 2026 when i was researching for orgs to contribute in operating systems i found linux org and some other orgs but haiku stood out.
Did you also apply to other GSoC organizations? If so, which is your order of preference?
Ans: No.
Estimated last day of classes/exams:
Ans: june 18
Estimated first day of classes:
Ans: august 3

Introduction

I’m pranav bisht, a 2nd year student in computer science at Dr. Akhilesh Das Gupta Institute of Professional Studies(India). I am fascinated by operating systems, compiler design and computer architecture as the amount of knowledge it takes to build one and also the sheer amount of things a operating system must do to just work is pretty mind blowing. My long term goal is to pursue a PhD in computer science with a focus on systems research.

The reason i choose haiku is mainly because it has a pretty clean code and has a great learning environment also its still in heavy development and i believe if i take a part in it i will be gaining knowledge far more that books can offer in a shorter time frame and it will all be practical knowledge and also it will help me solidify my technical understanding of things and help me better grasp some practical knowledge about system designs. The FreeBSD compatibility layer specifically caught my eye because i was into risc-v chips and was following bitluni a youtuber which has some pretty awesome projects on microcontroller based on risc-v architecture which is how i came to know about risc-v in the first place.

programming experience:
I have been daily driving linux (mostly Arch Linux and ubuntu) for over four years, so im very comfortable in unix like environments and command line toolchains. My primary languages are c/c++. I was fascinated by the Nand2Tetris curriculum and i have build a simple 16 function ALU from scratch. Im also heavily invested in a robotics society in my college and due to it have spent a lot of time working with microcontrollers and embedded systems.

Project Proposal

Topic - FreeBSD compatibility layer for FDT attached network devices.

Technical overview -

As haiku is actively being actively ported to arm and risc-v devices many of its drivers need to undergo changes for them to work on arm and risc-v devices due to architectural differences like -

- Instruction set difference -
  X86 devices uses CISC (Complex Instruction Set Computer) while ARM and RISC-V devices uses RISC (Reduced Instruction Set Computer).
- Memory access models -
  X86 has memory-to-registers operations meaning they can perform operations directly on memory locations, while ARM and RISC-V uses load-store models basically data must be loaded into registers before being processed by arithmetic instructions as only load and store instructions can access memory.
- Memory ordering -
  X86 has a strong memory model which guarantees strict ordering, while ARM and RISC-V use a weak model which allow reordering.
- I/O map -
  X86 uses a dedicated(can say its mixed as x86 can do mapped I/O too) I/O address space, whereas ARM and RISC-V uses Memory mapped I/O only.
- Endianness -
  X86 is strictly Little-endian, whereas ARM and RISC-V can be configured either Big-endian or Little-endian.

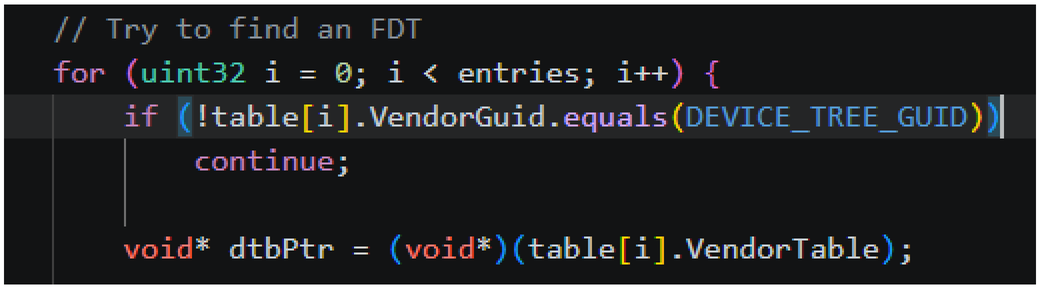
* Endianness is the order in which bytes are arranged in memory.

Current state of things -

Currently arm and risc-v port of haiku has very basic support for some driver that are native and no support for FreeBSD drivers that are ported recently to haiku os.

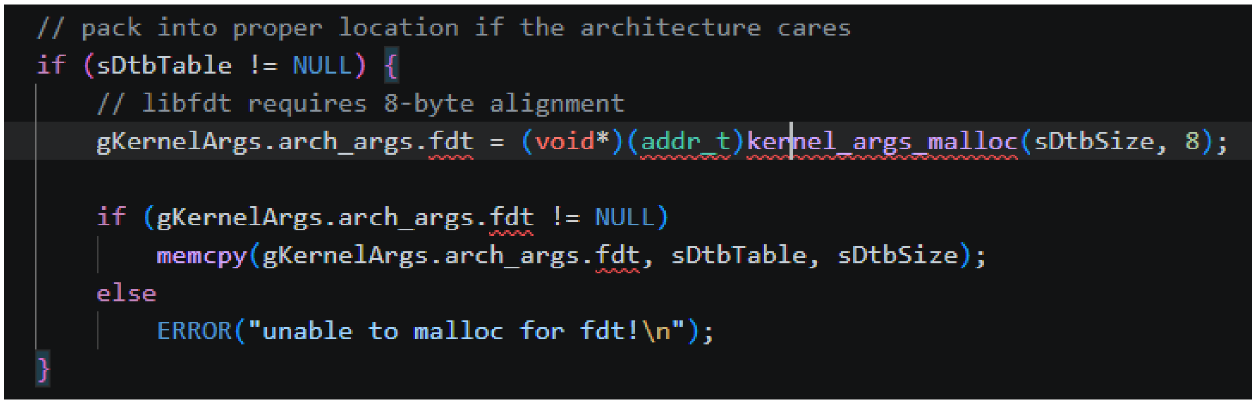
Currently haiku can -

- DTB parse - bootloader locates device tree and passes it to kernel. Libfdt is used.



```
// Try to find an FDT
for (uint32 i = 0; i < entries; i++) {
    if (!table[i].VendorGuid.equals(DEVICE_TREE_GUID))
        continue;

    void* dtbPtr = (void*)(table[i].VendorTable);
```

This locates the device tree and the next code snippet passes it to the kernel.



```
// pack into proper location if the architecture cares
if (sDtbTable != NULL) {
    // libfdt requires 8-byte alignment
    gKernelArgs.arch_args.fdt = (void*)(addr_t)kernel_args_malloc(sDtbSize, 8);

    if (gKernelArgs.arch_args.fdt != NULL)
        memcpy(gKernelArgs.arch_args.fdt, sDtbTable, sDtbSize);
    else
        ERROR("unable to malloc for fdt!\n");
}
```

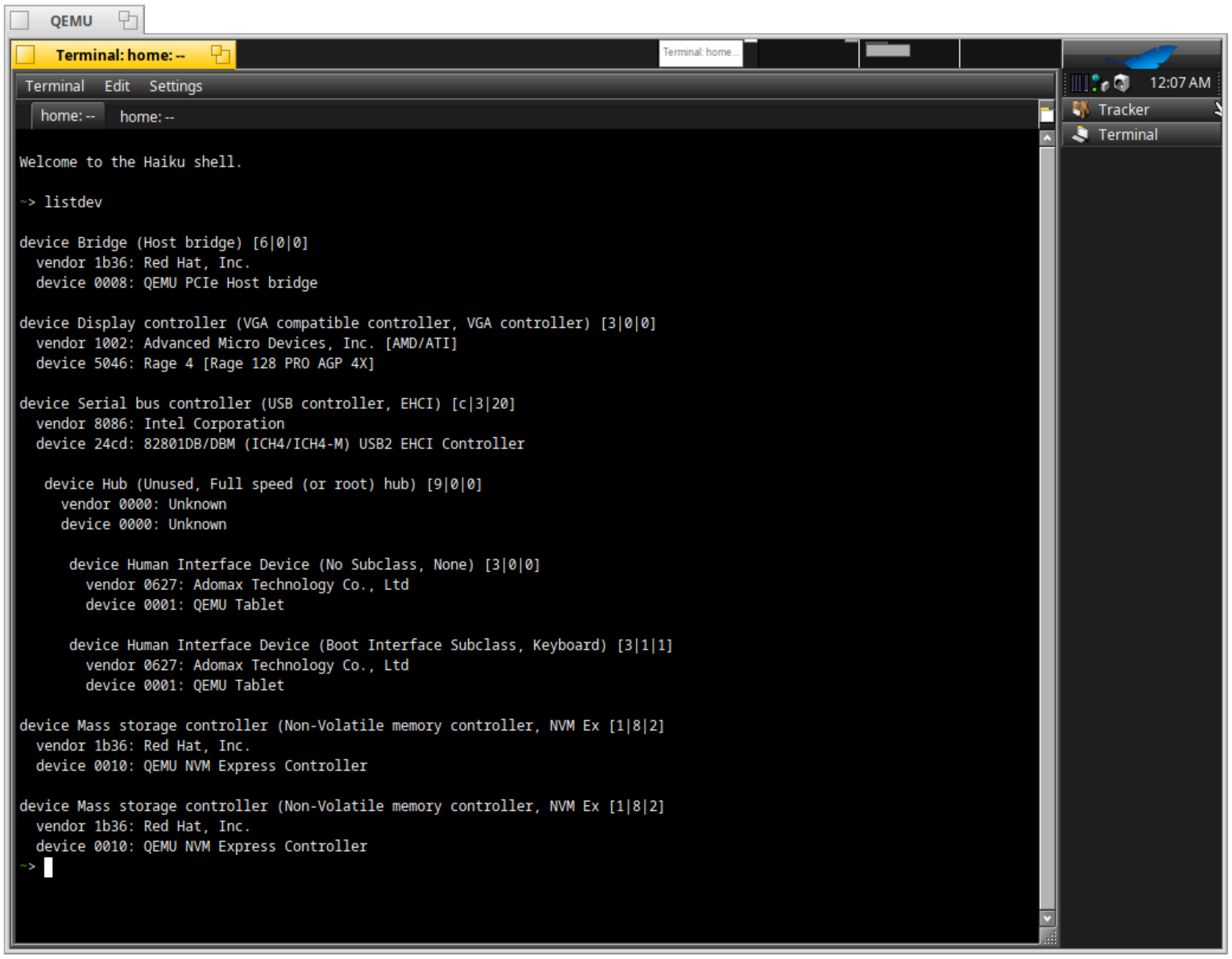
- The FDT bus manager can traverse the tree and create device_node in haiku's device tree.
- Interrupt controllers are supported for example GIC(Generic Interrupt Controller) in RISC-V and PLIC(Platform Level Interrupt Controller) in ARM.
- VirtIO support as most virtualization drivers works out of the box they attach to the FDT bus in QEMU.
- A basic UART.

Current drivers that are being loaded and that works are -

- NVME controller driver gets loaded but its using polling mode because its not receiving interrupts from the cpu meaning there is support for interrupt but its lacking.

```
nvme_disk: timed out waiting for interrupt!
nvme_disk: switching to polling mode, performance will be affected!
```

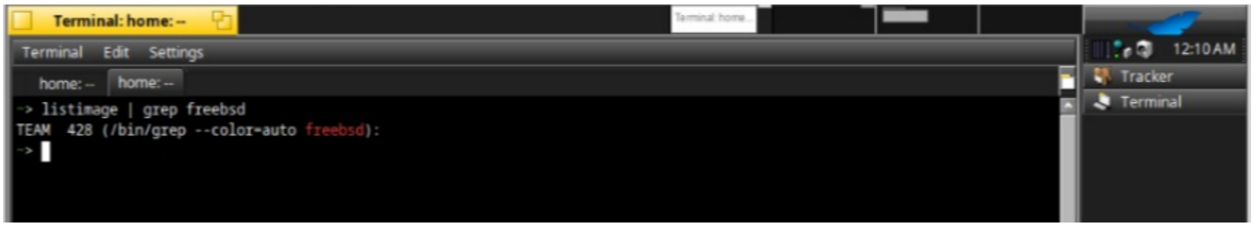
- VGA drivers are also loaded namely ATI driver but without acceleration only using Framebuffer.
- Usb_echi are also loaded allowing keyboard and mouse to be used.
- PCI bridge is also loaded for finding pci attached devices.



```
Welcome to the Haiku shell.

~> listdev

device Bridge (Host bridge) [6|0|0]
  vendor 1b36: Red Hat, Inc.
  device 0008: QEMU PCIe Host bridge

device Display controller (VGA compatible controller, VGA controller) [3|0|0]
  vendor 1002: Advanced Micro Devices, Inc. [AMD/ATI]
  device 5046: Rage 4 [Rage 128 PRO AGP 4X]

device Serial bus controller (USB controller, EHCI) [c|3|20]
  vendor 8086: Intel Corporation
  device 24cd: 82801DB/DBM (ICH4/ICH4-M) USB2 EHCI Controller

   device Hub (Unused, Full speed (or root) hub) [9|0|0]
     vendor 0000: Unknown
     device 0000: Unknown

      device Human Interface Device (No Subclass, None) [3|0|0]
        vendor 0627: Adomax Technology Co., Ltd
        device 0001: QEMU Tablet

      device Human Interface Device (Boot Interface Subclass, Keyboard) [3|1|1]
        vendor 0627: Adomax Technology Co., Ltd
        device 0001: QEMU Tablet

device Mass storage controller (Non-Volatile memory controller, NVM Ex [1|8|2]
  vendor 1b36: Red Hat, Inc.
  device 0010: QEMU NVM Express Controller

device Mass storage controller (Non-Volatile memory controller, NVM Ex [1|8|2]
  vendor 1b36: Red Hat, Inc.
  device 0010: QEMU NVM Express Controller
~>
```

Now, what doesn’t works -

- Networking on which haiku relies on FreeBSD driver are currently not supported at all.



```
~> listimage | grep freebsd
TEAM  428 (/bin/grep --color=auto freebsd):
~> 
```

- As ARM and RISC-V uses FDT, ACPI drivers are currently ignored.
- No HDA(high definition audio) as I2S drivers are currently not supported.

Goals -

- **Implement Interrupt Mapping** -
  Implement support for interrupt-map and interrupt-map-mask properties. Which will allow the Os to correctly route interrupts.

- **Clock and reset framework** -
  Create a clock manager within the FDT or bus or something similar that can handle clock signals and also resets to power on devices which are power off by default on ARM and RISC-V devices to save power. Currently this is handled by the bootloader.

- **FreeBSD Compatibility layer** -
  Map FDT to the device_t structure used by FreeBSD shim to allow open firmware calls to work with haiku's FDT bus.

Timeline -

- **Community bonding period (May/June) -**
  - Pick a specific FreeBSD driver to use as a test subject.
  - Further test currently working drivers for any flaws.

    - Explore FreeBSD documentation to better understand the working of its drivers.
    - Find examples and research how to implement clock/reset framework .

- **First month of coding (June/July) -**

    - Implement interrupt refactoring.
    - Goal - working interrupt parent lookup and interrupt map parsing.

- **Second month of coding (July/August) -**

    - Finalize the clock/reset framework.
    - Implement a basic Stub driver for QEMU.
    - Goal - able to provide clock and resets to devices.

- **Third month of coding (August/September) -**

    - Map the FDT to the FreeBSD shim.
    - Test it on physical hardware (raspberry pi 5)

- **After Google Summer of Code -**

    - Further work on the clock/reset framework.
    - Implement Userland GPIO api.
    - And some things i want to try are -
        - Can we use formal methods to prove a device tree parser is memory safe.
        - newer FPGA based SoC's have complex power management so can haiku handle like FDT overlays?

Expectations from Mentors -

I expect to run into undocumented edge case, especially regarding Haiku's clock manager and there are so many things that needs to be discussed. Hence, all i would need is pointers if i'm moving in the right direction because i know most of the people have jobs and everyone is doing this as a hobby but everyone is serious about things. So i will try to make the most out of the time i take out of your personal time.