Proposal for GSoC 2022

Haiku : ARM port and device tree support

About Me

BASIC INFO

Name: Zhihong Niu
Email: zone.niuzh@hotmail.com
GitHub Account: MRNIU
Time Zone: UTC+8
IRC Username: MRNIU
Trac username: PtrNull

EDUCATION

Xi'an Shiyou University, Xi'an China
Computer science , expected graduation July 2022

RESUME View my resume on github.

SUBMIT https://review.haiku-os.org/q/owner:Zone.Niuzh%2540hotmail.com

NOTE: YOU CAN SEE THE MARKDOWN EDITION HERE

Will you treat *Google Summer of Code* as full time employment?

No. I will volunteer during the summer vacation.

How many hours per week will you work?

30~40

List all obligations (and their dates) that may take time away from GSoC (a second job, vacations, classes, ...):

1st June~1st September	Summer vacation	
10th June~25st August	Voluntary Activity	4 hours a day except weekends

I can spend at least **4 hours a day on GSoC**, and if I have extra time, I wiil put it in.

Are you using *Google Summer of Code* to fullfill a university requirement -- internship, class credit, ..? (If so, you need to provide confirmation that your school approves. Remember, *Google Summer of Code* is a remote software development internship.)

No.

How and when did you first heard about Haiku?

via GSoC 2018

Did you also apply to other GSoC organizations? If so, which is your order of preference? (this has no impact on the selection process, we use this information only if you are selected as an intern by several of these organizations to decide which one would keep you)

No, Haiku is the **only** organization I apply for.

Estimated last day of classes/exams

25th May

Estimated first day of classes (for most interns, there will be some overlap with the GSoC coding period. This is fine, but we want to check that you have planned your timeline accordingly)

25th May is my last day at school.

Introduce yourself. (Who you are. Why you chose Haiku. What programming experience you have.)

See my resume here: [View my resume on github](#).

In 2018, I tried to apply for haiku, but was rejected due to lack of knowledge. This year is my last year in school, and I hope to draw a perfect end to my college life.

Project Proposal. (Title. Description. Goals.)

Annex

Timeline. Include what you plan to have accomplished by the end of:

See Proposal.

Expectations from Mentors. (What do you expect Haiku's mentors to help you with?)

Most of the problems can be solved by reading the code, but knowing more about how the author designed concrete code before that will help me to be efficient.

Project Name

HaiKu: ARM port and device tree support

Description

Under x86, the address of most hardware is fixed, but for architectures such as ARM/RISCV/MIPS, the address of the device varies according to the vendor.

To solve the problem of device address, the concept of device tree was introduced.

The human-readable format of the device tree is.dts, the bootloader passes dtb info to the kernel at system startup.

The bootloader passes dtb info encoded according to the [specification](https://buildmedia.readthedocs.org/media/pdf/devicetree-specification/latest/devicetree-specification.pdf) to the kernel at system startup, and kernel can access the device after parsing the device information according to the specification.

Goals

1. Run Haiku on ARM via qemu, -machine for raspi2b. The hardware will be tested if I have extra time.
2. Supports at least one type of large-capacity storage device via the device tree.

The first thing I wanna do is virtio_disk, X512 has done a lot for virtio, It will familiarize me with device/driver's design.

After that I will try SD-MMC. I used to write code related to sd card.

By the way, I am not sure if I can finish both of them within GSoC, but at least there is no problem getting virtio_disk, and if SD-MMC does not get done in time, I will finish it after GSoC. So I hope I can pass the evaluation by completing one of the above goals before the final.

TimeLine

Community Bonding period	May 20 - June 12	Read the code, refine the plan, and identify problems early
Coding Phase I June 13 - July 29	Week 1-2	Check existing code and fix any bugs. Get and printing out the contents of DTS
	Week 3-6	Virtio_disk Support

	Week 7	Prepare for First Evaluation
Coding Phase II July 30 - September 4	Week 8-10	SD-MMC Support
	Week 11-13	Buffer Debug and fix bugs
Final Week	September 5 - September 12	Prepare for Final Evaluation
After GSoC		I am interested in the design and implementation of Haiku. If there is a problem when reading the code, I will create an issue or directly modify it.

Boot

I've combed through haiku's startup logic so far

1. src/system/boot/platform/efi/start.cpp `extern "C" efi_status efi_main(efi_handle image, efi_system_table *systemTable)`

   EFI entry

   Global object construction

   Console Initialization

   Serial Port Initialization

   CPU Initialization

   ACPI Initialization

   dtb Initialization

   timer Initialization

   SMP Initialization

   Call `main`

2. src/system/boot/loader/main.cpp `extern "C" int main(stage2_args *args)`

   Set the base operating environment for kernel: heap, vfs, find kernel.

   load kernel and setup kernel args.

3. src/system/boot/platform/efi/start.cpp `extern "C" void platform_start_kernel(void)`

get kernel entry

init other core

MMU Initialization

remap kernel args

setup kernel stack

4. src/system/boot/platform/efi/arch/arm/arch_start.cpp `void arch_start_kernel(addr_t kernelEntry)`

map kernel

Call 'enter_kernel', which is a function pointer obtained by 'get_kernel_entry' of 'platform_start_kernel'

5. src/system/boot/platform/efi/arch/arm/entry.S `FUNCTION(arch_enter_kernel)`

This is a piece of assembler

Pass the argument and jump to the kernel entry address of 'get_kernel_entry' passed in 'platform_start_kernel', which is resolved from the ELF file.

The bootloader ends up here.

6. src/system/kernel/main.cpp `extern "C" int _start(kernel_args *bootKernelArgs, int currentCPU)`

Kernel entry, finally we get into the kernel, same as `src/system/ldscripts/arm/boot_loader_efi.ld`

`arch_platform_init` setup `gFDT`, which is what we need to use in this project.

At present, there are still problems(below) with the startup of ARM, and I will continue the investigation. After that, the main work is on the analysis of DTS and the support of device drivers.

DTB and Drivers

About dtb, I've written some code for that, I put it below. The whole code can be found at https://github.com/MRNIU/SimpleKernel/blob/master/src/drv/dtb/dtb.cpp, it's a simple DTB parser program.

For Drivers, it should be based on X512's work.
fdt_device_module_info is the most basic part, on this basis we got *virtio_mmio* bus, and then our virtio_blk device can be accessed by mode *VirtioDevice.*

DTB parsing

```cpp

/**
 * @file dtb.cpp
 * @brief dtb 解析实现
 * @author Zone.N (Zone.Niuzh@hotmail.com)
 * @version 1.0
 * @date 2021-09-18
 * @copyright MIT LICENSE
 * https://github.com/Simple-XX/SimpleKernel
 * Based on https://github.com/brenns10/sos
 * @par change log:
 * <table>
 * <tr><th>Date<th>Author<th>Description
 * <tr><td>2021-09-18<td>digmouse233<td>迁移到 doxygen
 * </table>
 */

#include "stdint.h"
#include "stdio.h"
#include "endian.h"
#include "assert.h"
#include "common.h"
#include "boot_info.h"
#include "resource.h"
#include "dtb.h"

// 所有节点
DTB::node_t DTB::nodes[MAX_NODES];
// 节点数
size_t DTB::node_t::count = 0;
// 所有 phandle
DTB::phandle_map_t DTB::phandle_map[MAX_NODES];
// phandle 数
size_t DTB::phandle_map_t::count = 0;

bool DTB::path_t::operator==(const DTB::path_t *_path) {
    if (len != _path->len) {
```

```
        return false;
    }
    for (size_t i = 0; i < len; i++) {
        if (strcmp(path[i], _path->path[i]) != 0) {
            return false;
        }
    }
    return true;
}

bool DTB::path_t::operator==(const char *_path) {
    // 路径必须以 '/' 开始
    if (_path[0] != '/') {
        return false;
    }
    // 记录当前 _path 处理到的下标
    size_t tmp = 0;
    for (size_t i = 0; i < len; i++) {
        if (strncmp(path[i], &_path[tmp + i], strlen(path[i])) != 0) {
            return false;
        }
        tmp += strlen(path[i]);
    }
    return true;
}

DTB::node_t *DTB::get_phandle(uint32_t _phandle) {
    // 在 phandle_map 中寻找对应的节点
    for (size_t i = 0; i < phandle_map[0].count; i++) {
        if (phandle_map[i].phandle == _phandle) {
            return phandle_map[i].node;
        }
    }
    return nullptr;
}

DTB::dt_fmt_t DTB::get_fmt(const char *_prop_name) {
    // 默认为 FMT_UNKNOWN
    dt_fmt_t res = FMT_UNKNOWN;
    for (size_t i = 0; i < sizeof(props) / sizeof(dt_prop_fmt_t); i++) {
        // 找到了则更新
        if (strcmp(_prop_name, props[i].prop_name) == 0) {
            res = props[i].fmt;
        }
    }
    return res;
```

```
}

void DTB::print_attr_propenc(const iter_data_t *_iter, size_t *_cells,
                             size_t _len) {
    // 字节总数
    uint32_t entry_size = 0;
    // 属性长度
    uint32_t remain = _iter->prop_len;
    // 属性数据
    uint32_t *reg = _iter->prop_addr;
    printf("%s: ", _iter->prop_name);

    // 计算
    for (size_t i = 0; i < _len; i++) {
        entry_size += 4 * _cells[i];
    }

    printf("(len=%u/%u) ", _iter->prop_len, entry_size);

    // 理论上应该是可以整除的，如果不行说明有问题
    assert(_iter->prop_len % entry_size == 0);

    // 数据长度大于 0
    while (remain > 0) {
        std::cout << "<";
        for (size_t i = 0; i < _len; i++) {
            // 直接输出
            for (size_t j = 0; j < _cells[i]; j++) {
                printf("0x%X ", be32toh(*reg));
                // 下一个 cell
                reg++;
                // 减 4, 即一个 cell 大小
                remain -= 4;
            }

            if (i != _len - 1) {
                std::cout << "| ";
            }
        }
        // \b 删除最后一个 "| " 中的空格
        std::cout << "\b>";
    }
    return;
}

void DTB::fill_resource(resource_t *_resource, const node_t *_node,
```

```
                    const prop_t *_prop) {
    // 如果 _resource 名称为空则使用 _node 路径填充
    if (_resource->name == nullptr) {
        _resource->name = _node->path.path[_node->path.len - 1];
    }
    // 内存类型
    if ((_resource->type & resource_t::MEM) && (_resource->mem.len == 0)) {
        // 根据 address_cells 与 size_cells 填充
        // resource 一般来说两者是相等的
        if (_node->parent->address_cells == 1) {
            assert(_node->parent->size_cells == 1);
            _resource->mem.addr = be32toh(((uint32_t *)_prop->addr)[0]);
            _resource->mem.len  = be32toh(((uint32_t *)_prop->addr)[1]);
        }
        else if (_node->parent->address_cells == 2) {
            assert(_node->parent->size_cells == 2);
            _resource->mem.addr = be32toh(((uint32_t *)_prop->addr)[0]) +
                                  be32toh(((uint32_t *)_prop->addr)[1]);
            _resource->mem.len = be32toh(((uint32_t *)_prop->addr)[2]) +
                                 be32toh(((uint32_t *)_prop->addr)[3]);
        }
        else {
            assert(0);
        }
    }
    else if (_resource->type & resource_t::INTR_NO) {
        _resource->intr_no = be32toh(((uint32_t *)_prop->addr)[0]);
    }
    return;
}

DTB::node_t *DTB::find_node_via_path(const char *_path) {
    node_t *res = nullptr;
    // 遍历 nodes
    for (size_t i = 0; i < nodes[0].count; i++) {
        // 如果 nodes[i] 中有属性/值对符合要求
        if (nodes[i].path == _path) {
            // 设置返回值
            res = &nodes[i];
        }
    }
    return res;
}

void DTB::dtb_mem_reserved(void) {
    fdt_reserve_entry_t *entry = dtb_info.reserved;
```

```
    if (entry->addr_le || entry->size_le) {
        // 目前没有考虑这种情况，先报错
        assert(0);
    }
    return;
}

void DTB::dtb_iter(uint8_t _cb_flags, bool (*_cb)(const iter_data_t *, void *),
                   void *_data, uintptr_t _addr) {
    // 迭代变量
    iter_data_t iter;
    // 路径深度
    iter.path.len = 0;
    // 数据地址
    iter.addr = (uint32_t *)_addr;
    // 节点索引
    iter.nodes_idx = 0;
    // 开始 flag
    bool begin = true;

    while (1) {
        //
        iter.type = be32toh(iter.addr[0]);
        switch (iter.type) {
            case FDT_NOP: {
                // 跳过 type
                iter.addr++;
                break;
            }
            case FDT_BEGIN_NODE: {
                // 第 len 深底的名称
                iter.path.path[iter.path.len] = (char *)(iter.addr + 1);
                // 深度+1
                iter.path.len++;
                iter.nodes_idx = begin ? 0 : (iter.nodes_idx + 1);
                begin          = false;
                if (_cb_flags & DT_ITER_BEGIN_NODE) {
                    if (_cb(&iter, _data)) {
                        return;
                    }
                }
                // 跳过 type
                iter.addr++;
                // 跳过 name
                iter.addr +=
                    COMMON::ALIGN(strlen((char *)iter.addr) + 1, 4) / 4;
```

```
            break;
        }
        case FDT_END_NODE: {
            if (_cb_flags & DT_ITER_END_NODE) {
                if (_cb(&iter, _data)) {
                    return;
                }
            }
            // 这一级结束了，所以 -1
            iter.path.len--;
            // 跳过 type
            iter.addr++;
            break;
        }
        case FDT_PROP: {
            iter.prop_len  = be32toh(iter.addr[1]);
            iter.prop_name = (char *)(dtb_info.str + be32toh(iter.addr[2]));
            iter.prop_addr = iter.addr + 3;
            if (_cb_flags & DT_ITER_PROP) {
                if (_cb(&iter, _data)) {
                    return;
                }
            }
            iter.prop_name = nullptr;
            iter.prop_addr = nullptr;
            // 跳过 type
            iter.addr++;
            // 跳过 len
            iter.addr++;
            // 跳过 nameoff
            iter.addr++;
            // 跳过 data，并进行对齐
            iter.addr += COMMON::ALIGN(iter.prop_len, 4) / 4;
            iter.prop_len = 0;
            break;
        }
        case FDT_END: {
            return;
        }
        default: {
            printf("unrecognized token 0x%X\n", iter.type);
            return;
        }
    }
}
return;
```

```
}

DTB::dtb_info_t DTB::dtb_info;

/*
 * This callback constructs tracking information about each node.
 */
bool DTB::dtb_init_cb(const iter_data_t *_iter, void *) {
    // 索引
    size_t idx = _iter->nodes_idx;
    // 根据类型
    switch (_iter->type) {
        // 开始
        case FDT_BEGIN_NODE: {
            // 设置节点基本信息
            nodes[idx].path  = _iter->path;
            nodes[idx].addr  = _iter->addr;
            nodes[idx].depth = _iter->path.len;
            // 设置默认值
            nodes[idx].address_cells    = 2;
            nodes[idx].size_cells       = 2;
            nodes[idx].interrupt_cells  = 0;
            nodes[idx].phandle          = 0;
            nodes[idx].interrupt_parent = nullptr;
            // 设置父节点
            // 如果不是根节点
            if (idx != 0) {
                size_t i = idx - 1;
                while (nodes[i].depth != nodes[idx].depth - 1) {
                    i--;
                }
                nodes[idx].parent = &nodes[i];
            }
            // 索引为 0 说明是根节点
            else {
                // 根节点的父节点为空
                nodes[idx].parent = nullptr;
            }
            break;
        }
        case FDT_PROP: {
            // 获取 cells 信息
            if (strcmp(_iter->prop_name, "#address-cells") == 0) {
                nodes[idx].address_cells = be32toh(_iter->addr[3]);
            }
            else if (strcmp(_iter->prop_name, "#size-cells") == 0) {
```

```
            nodes[idx].size_cells = be32toh(_iter->addr[3]);
        }
        else if (strcmp(_iter->prop_name, "#interrupt-cells") == 0) {
            nodes[idx].interrupt_cells = be32toh(_iter->addr[3]);
        }
        // phandle 信息
        else if (strcmp(_iter->prop_name, "phandle") == 0) {
            nodes[idx].phandle = be32toh(_iter->addr[3]);
            // 更新 phandle_map
            phandle_map[phandle_map[0].count].phandle = nodes[idx].phandle;
            phandle_map[phandle_map[0].count].node    = &nodes[idx];
            phandle_map[0].count++;
        }
        // 添加属性
        nodes[idx].props[nodes[idx].prop_count].name = _iter->prop_name;
        nodes[idx].props[nodes[idx].prop_count].addr =
            (uintptr_t)(_iter->addr + 3);
        nodes[idx].props[nodes[idx].prop_count].len =
            be32toh(_iter->addr[1]);
        nodes[idx].prop_count++;
        break;
      }
      case FDT_END_NODE: {
        // 节点数+1
        nodes[0].count = idx + 1;
        break;
      }
    }
    // 返回 false 表示需要迭代全部节点
    return false;
}

bool DTB::dtb_init_interrupt_cb(const iter_data_t *_iter, void *) {
    uint8_t  idx = _iter->nodes_idx;
    uint32_t phandle;
    node_t  *parent;
    // 设置中断父节点
    if (strcmp(_iter->prop_name, "interrupt-parent") == 0) {
        phandle = be32toh(_iter->addr[3]);
        parent  = get_instance().get_phandle(phandle);
        // 没有找到则报错
        assert(parent != nullptr);
        nodes[idx].interrupt_parent = parent;
    }
    // 返回 false 表示需要迭代全部节点
    return false;
```

```
}

DTB &DTB::get_instance(void) {
    /// 定义全局 DTB 对象
    static DTB dtb;
    return dtb;
}

bool DTB::dtb_init(void) {
    // 头信息
    dtb_info.header = (fdt_header_t *)BOOT_INFO::boot_info_addr;
    // 魔数
    assert(be32toh(dtb_info.header->magic) == FDT_MAGIC);
    // 版本
    assert(be32toh(dtb_info.header->version) == FDT_VERSION);
    // 设置大小
    BOOT_INFO::boot_info_size = be32toh(dtb_info.header->totalsize);
    // 内存保留区
    dtb_info.reserved =
        (fdt_reserve_entry_t *)(BOOT_INFO::boot_info_addr +
                                be32toh(dtb_info.header->off_mem_rsvmap));
    // 数据区
    dtb_info.data =
        BOOT_INFO::boot_info_addr + be32toh(dtb_info.header->off_dt_struct);
    // 字符区
    dtb_info.str =
        BOOT_INFO::boot_info_addr + be32toh(dtb_info.header->off_dt_strings);
    // 检查保留内存
    dtb_mem_reserved();
    // 初始化 map
    bzero(nodes, sizeof(nodes));
    bzero(phandle_map, sizeof(phandle_map));
    // 初始化节点的基本信息
    dtb_iter(DT_ITER_BEGIN_NODE | DT_ITER_END_NODE | DT_ITER_PROP,
dtb_init_cb,
             nullptr);
    // 中断信息初始化，因为需要查找 phandle，所以在基本信息初始化完成后进行
    dtb_iter(DT_ITER_PROP, dtb_init_interrupt_cb, nullptr);
// #define DEBUG
#ifdef DEBUG
    // 输出所有信息
    for (size_t i = 0; i < nodes[0].count; i++) {
        std::cout << nodes[i].path << ": " << std::endl;
        for (size_t j = 0; j < nodes[i].prop_count; j++) {
            printf("%s: ", nodes[i].props[j].name);
            for (size_t k = 0; k < nodes[i].props[j].len / 4; k++) {
```

```
                printf("0x%X ",
                       be32toh(((uint32_t *)nodes[i].props[j].addr)[k]));
            }
            printf("\n");
        }
    }
#undef DEBUG
#endif
    return true;
}

/// @todo 这里看起来似乎可以优化
bool DTB::find_via_path(const char *_path, resource_t *_resource) {
    // 找到节点
    auto node = find_node_via_path(_path);
    // std::cout << node->path << std::endl;
    // 找到 reg
    for (size_t i = 0; i < node->prop_count; i++) {
        // printf("node->props[i].name: %s\n", node->props[i].name);
        if (strcmp(node->props[i].name, "reg") == 0) {
            // 填充数据
            _resource->type |= resource_t::MEM;
            fill_resource(_resource, node, &node->props[i]);
        }
        else if (strcmp(node->props[i].name, "interrupts") == 0) {
            // 填充数据
            _resource->type |= resource_t::INTR_NO;
            fill_resource(_resource, node, &node->props[i]);
        }
    }
    return true;
}

/// @todo 这里看起来似乎可以优化
size_t DTB::find_via_prefix(const char *_prefix, resource_t *_resource) {
    size_t res = 0;
    // 遍历所有节点，查找
    // 由于 @ 均为最底层节点，所以直接比较最后一级即可
    for (size_t i = 0; i < nodes[0].count; i++) {
        if (strncmp(nodes[i].path.path[nodes[i].path.len - 1], _prefix,
                    strlen(_prefix)) == 0) {
            // 找到 reg
            for (size_t j = 0; j < nodes[i].prop_count; j++) {
                if (strcmp(nodes[i].props[j].name, "reg") == 0) {
                    _resource[res].type |= resource_t::MEM;
                    // 填充数据
```

```
                fill_resource(&_resource[res], &nodes[i],
                        &nodes[i].props[j]);
            }
            else if (strcmp(nodes[i].props[j].name, "interrupts") == 0) {
                _resource[res].type |= resource_t::INTR_NO;
                // 填充数据
                fill_resource(&_resource[res], &nodes[i],
                        &nodes[i].props[j]);
            }
        }
        res++;
    }
  }
  return res;
}

std::ostream &operator<<(std::ostream &_os, const DTB::iter_data_t &_iter) {
  // 输出路径
  _os << _iter.path << ": ";
  // 根据属性类型输出
  switch (DTB::get_instance().get_fmt(_iter.prop_name)) {
    // 未知
    case DTB::FMT_UNKNOWN: {
      warn("%s: (unknown format, len=0x%X)", _iter.prop_name,
          _iter.prop_len);
      break;
    }
    // 空
    case DTB::FMT_EMPTY: {
      _os << _iter.prop_name << ": (empty)";
      break;
    }
    // 32 位整数
    case DTB::FMT_U32: {
      printf("%s: 0x%X", _iter.prop_name,
          be32toh(*(uint32_t *)_iter.prop_addr));
      break;
    }
    // 64 位整数
    case DTB::FMT_U64: {
      printf("%s: 0x%p", _iter.prop_name,
          be32toh(*(uint64_t *)_iter.prop_addr));
      break;
    }
    // 字符串
    case DTB::FMT_STRING: {
```

```
        _os << _iter.prop_name << ": " << (char *)_iter.prop_addr;
        break;
    }
    // phandle
    case DTB::FMT_PHANDLE: {
        uint32_t     phandle = be32toh(_iter.addr[3]);
        DTB::node_t *ref     = DTB::get_instance().get_phandle(phandle);
        if (ref != nullptr) {
            printf("%s: <phandle &%s>", _iter.prop_name, ref->path.path[0]);
        }
        else {
            printf("%s: <phandle 0x%X>", _iter.prop_name, phandle);
        }
        break;
    }
    // 字符串列表
    case DTB::FMT_STRINGLIST: {
        size_t len = 0;
        char  *str = (char *)_iter.prop_addr;
        _os << _iter.prop_name << ": [";
        while (len < _iter.prop_len) {
            // 用 "" 分隔
            _os << "\"" << str << "\"";
            len += strlen(str) + 1;
            str = (char *)((uint8_t *)_iter.prop_addr + len);
        }
        _os << "]";
        break;
    }
    // reg, 不定长的 32 位数据
    case DTB::FMT_REG: {
        // 获取节点索引
        uint8_t idx = _iter.nodes_idx;
        // 全部 cells 大小
        // devicetree-specification-v0.3.pdf#2.3.6
        size_t cells[] = {
            DTB::nodes[idx].parent->address_cells,
            DTB::nodes[idx].parent->size_cells,
        };
        // 调用辅助函数进行输出
        DTB::get_instance().print_attr_propenc(
            &_iter, cells, sizeof(cells) / sizeof(size_t));
        break;
    }
    case DTB::FMT_RANGES: {
        // 获取节点索引
```

```
            uint8_t idx = _iter.nodes_idx;
            // 全部 cells 大小
            // devicetree-specification-v0.3.pdf#2.3.8
            size_t cells[] = {
                DTB::nodes[idx].address_cells,
                DTB::nodes[idx].parent->address_cells,
                DTB::nodes[idx].size_cells,
            };
            // 调用辅助函数进行输出
            DTB::get_instance().print_attr_propenc(
                &_iter, cells, sizeof(cells) / sizeof(size_t));
            break;
        }
        default: {
            assert(0);
            break;
        }
    }
    return _os;
}

std::ostream &operator<<(std::ostream &_os, const DTB::path_t &_path) {
    if (_path.len == 1) {
        _os << "/";
    }
    for (size_t i = 1; i < _path.len; i++) {
        _os << "/";
        _os << _path.path[i];
    }
    return _os;
}

namespace BOOT_INFO {
// 地址
uintptr_t boot_info_addr;
// 长度
size_t boot_info_size;
// 启动核
size_t dtb_init_hart;

bool inited = false;

bool init(void) {
    auto res = DTB::get_instance().dtb_init();
    if (inited == false) {
        inited = true;
```

```
        info("BOOT_INFO init.\n");
    }
    else {
        info("BOOT_INFO reinit.\n");
    }
    return res;
}

resource_t get_memory(void) {
    resource_t resource;
    // 获取资源信息
    assert(DTB::get_instance().find_via_prefix("memory@", &resource) == 1);
    return resource;
}

size_t find_via_prefix(const char *_prefix, resource_t *_resource) {
    return DTB::get_instance().find_via_prefix(_prefix, _resource);
}

resource_t get_clint(void) {
    resource_t resource;
    // 设置 resource 基本信息
    resource.type = resource_t::MEM;
    assert(DTB::get_instance().find_via_prefix("clint@", &resource) == 1);
    return resource;
}

resource_t get_plic(void) {
    resource_t resource;
    // 设置 resource 基本信息
    resource.type = resource_t::MEM;
    assert(DTB::get_instance().find_via_prefix("plic@", &resource) == 1);
    return resource;
}
}; // namespace BOOT_INFO

```

The current ARM with qemu

```
> qemu-system-arm -bios u-boot.bin -M virt -cpu cortex-a15 -m 2048 \
    -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 \
```

```
    -drive file="haiku-mmc.image",if=none,format=raw,id=x0 \
    -device ramfb -usb -device qemu-xhci,id=xhci -device usb-mouse -device usb-kbd
-serial stdio

  U-Boot 2021.01-rc4-00029-gab865a8ee5 (Dec 28 2020 - 20:37:22 -0600)

  DRAM:  2 GiB
  Flash: 128 MiB
  *** Warning - bad CRC, using default environment

  In:    pl011@9000000
  Out:   pl011@9000000
  Err:   pl011@9000000
  Net:   No ethernet found.
  Hit any key to stop autoboot:  0
  starting USB...
  No working controllers found
  USB is stopped. Please issue 'usb start' first.
  scanning bus for devices...

  Device 0: unknown device

  Device 0: QEMU VirtIO Block Device
            Type: Hard Disk
            Capacity: 351.9 MB = 0.3 GB (720886 x 512)
  ... is now current device
  Scanning virtio 0:1...
  Found U-Boot script /boot.scr
  1384 bytes read in 2 ms (675.8 KiB/s)
  ## Executing script at 40200000
  Haiku u-boot script entry
  50 bytes read in 3 ms (15.6 KiB/s)
  uEnv.txt says to look for efi bootloader named EFI/BOOT/BOOTARM.EFI on virtio 0!
  Found EFI/BOOT/BOOTARM.EFI on virtio 0!
  Loading bootloader...
  492034 bytes read in 2 ms (234.6 MiB/s)
  Using internal DTB...
  Launching EFI loader...
  Scanning disk virtio-blk#0...
  ** Unrecognized filesystem type **
  Found 3 disks
  Missing RNG device for EFI_RNG_PROTOCOL
  ** Unable to read file ubootefi.var **
  Failed to load EFI variables
  Booting /EFI\BOOT\BOOTARM.EFI
  arch_smp_register_cpu()
```

```
  cpu
    id: 0
  GOP protocol not found
  Welcome to the Haiku boot loader!
  add_partitions_for(0xbbd93138, mountFS = no)
  add_partitions_for(fd = 0, mountFS = no)
  0xbbd93178 Partition::Partition
  0xbbd93178 Partition::Scan()
  check for partitioning_system: GUID Partition Map
  ReadAt: blockIo error reading from device!
  check for partitioning_system: Intel Partition Map
    priority: 810
  check for partitioning_system: Intel Extended Partition
  0xbbd932f8 Partition::Partition
  0xbbd93178 Partition::AddChild 0xbbd932f8
  0xbbd932f8 Partition::SetParent 0xbbd93178
  new child partition!
  0xbbd933c0 Partition::Partition
  0xbbd93178 Partition::AddChild 0xbbd933c0
  0xbbd933c0 Partition::SetParent 0xbbd93178
  new child partition!
  0xbbd93178 Partition::Scan(): scan child 0xbbd932f8 (start = 20966400, size = 33554432,
parent = 0xbbd93178)!
  0xbbd932f8 Partition::Scan()
  check for partitioning_system: GUID Partition Map
  check for partitioning_system: Intel Partition Map
  check for partitioning_system: Intel Extended Partition
  0xbbd93178 Partition::Scan(): scan child 0xbbd933c0 (start = 54520832, size =
314572800, parent = 0xbbd93178)!
  0xbbd933c0 Partition::Scan()
  check for partitioning_system: GUID Partition Map
  check for partitioning_system: Intel Partition Map
  check for partitioning_system: Intel Extended Partition
  0xbbd93178 Partition::~Partition
  0xbbd932f8 Partition::SetParent 0x00000000
  0xbbd933c0 Partition::SetParent 0x00000000
  0xbbd932f8 Partition::_Mount check for file_system: BFS Filesystem
  0xbbd932f8 Partition::_Mount check for file_system: FAT32 Filesystem
  0xbbd932f8 Partition::_Mount check for file_system: TAR Filesystem
  0xbbd932f8 Partition::~Partition
  0xbbd933c0 Partition::_Mount check for file_system: BFS Filesystem
  PackageVolumeInfo::SetTo()
  PackageVolumeInfo::_InitState(): failed to parse activated-packages: No such file or
directory
  load kernel kernel_arm...
  maximum boot loader heap usage: 345608, currently used: 337248
```

```
  Chosen UART:
    kind: pl011
    regs: 0x9000000, 0x1000
    irq: 1
    clock: 24000000
  Chosen interrupt controller:
    kind: gicv2
    regs: 0x8000000, 0x10000
          0x8010000, 0x10000
  kernel:
    text: 0x80000000, 0x193000
    data: 0x801a3000, 0x4a000
    entry: 0x8006f408
  Kernel stack at 0x8241b000
  System provided memory map:
    phys: 0x40000000-0x47f00000, virt: 0x40000000-0x47f00000, type:
ConventionalMemory (0x7), attr: 0x8
    phys: 0x47f00000-0x48003000, virt: 0x47f00000-0x48003000, type:
ACPIReclaimMemory (0x9), attr: 0x8
    phys: 0x48003000-0x7fffe000, virt: 0x48003000-0x7fffe000, type: ConventionalMemory
(0x7), attr: 0x8
    phys: 0x7fffe000-0x7ffff000, virt: 0x7fffe000-0x7ffff000, type: LoaderData (0x2), attr: 0x8
    phys: 0x7ffff000-0xbb786000, virt: 0x7ffff000-0xbb786000, type: ConventionalMemory
(0x7), attr: 0x8
    phys: 0xbb786000-0xbdd93000, virt: 0xbb786000-0xbdd93000, type: LoaderData (0x2),
attr: 0x8
    phys: 0xbdd93000-0xbddf2000, virt: 0xbdd93000-0xbddf2000, type: LoaderCode (0x1),
attr: 0x8
    phys: 0xbddf2000-0xbddf6000, virt: 0xbddf2000-0xbddf6000, type:
ReservedMemoryType (0x0), attr: 0x8
    phys: 0xbddf6000-0xbddf7000, virt: 0xbddf6000-0xbddf7000, type: BootServicesData
(0x4), attr: 0x8
    phys: 0xbddf7000-0xbddf8000, virt: 0xbddf7000-0xbddf8000, type: RuntimeServicesData
(0x6), attr: 0x8000000000000008
    phys: 0xbddf8000-0xbddfa000, virt: 0xbddf8000-0xbddfa000, type: BootServicesData
(0x4), attr: 0x8
    phys: 0xbddfa000-0xbddfb000, virt: 0xbddfa000-0xbddfb000, type:
ReservedMemoryType (0x0), attr: 0x8
    phys: 0xbddfb000-0xbddfe000, virt: 0xbddfb000-0xbddfe000, type: RuntimeServicesData
(0x6), attr: 0x8000000000000008
    phys: 0xbddfe000-0xbddff000, virt: 0xbddfe000-0xbddff000, type: BootServicesData
(0x4), attr: 0x8
    phys: 0xbddff000-0xbde03000, virt: 0xbddff000-0xbde03000, type: RuntimeServicesData
(0x6), attr: 0x8000000000000008
    phys: 0xbde03000-0xbde04000, virt: 0xbde03000-0xbde04000, type:
ReservedMemoryType (0x0), attr: 0x8
```

phys: 0xbde04000-0xbde05000, virt: 0xbde04000-0xbde05000, type: BootServicesData (0x4), attr: 0x8
phys: 0xbde05000-0xbde06000, virt: 0xbde05000-0xbde06000, type: ReservedMemoryType (0x0), attr: 0x8
phys: 0xbde06000-0xbde08000, virt: 0xbde06000-0xbde08000, type: BootServicesData (0x4), attr: 0x8
phys: 0xbde08000-0xbde09000, virt: 0xbde08000-0xbde09000, type: ReservedMemoryType (0x0), attr: 0x8
phys: 0xbde09000-0xbde0a000, virt: 0xbde09000-0xbde0a000, type: BootServicesData (0x4), attr: 0x8
phys: 0xbde0a000-0xbde0e000, virt: 0xbde0a000-0xbde0e000, type: ReservedMemoryType (0x0), attr: 0x8
phys: 0xbde0e000-0xbde0f000, virt: 0xbde0e000-0xbde0f000, type: BootServicesData (0x4), attr: 0x8
phys: 0xbde0f000-0xbde10000, virt: 0xbde0f000-0xbde10000, type: ReservedMemoryType (0x0), attr: 0x8
phys: 0xbde10000-0xbff51000, virt: 0xbde10000-0xbff51000, type: LoaderData (0x2), attr: 0x8
phys: 0xbff51000-0xbff53000, virt: 0xbff51000-0xbff53000, type: RuntimeServicesCode (0x5), attr: 0x8000000000000008
phys: 0xbff53000-0xc0000000, virt: 0xbff53000-0xc0000000, type: LoaderData (0x2), attr: 0x8
Calling ExitBootServices. So long, EFI!
Switched to legacy serial output
enter_kernel(ttbr0: 0xbb750000, kernelArgs: 0x8241f000, kernelEntry: 0x8006f408, sp: 0x8241f000)
Welcome to kernel debugger output!
Haiku revision: hrev56013+2+g003f0b+dirty, debug level: 2
mark_page_range_in_use(0x0, 0x40000): start page is before free list
Enabled high vectors
status_t arch_vm_set_memory_type(VMArea*, phys_addr_t, uint32): undefined type 10000000!
status_t arch_vm_set_memory_type(VMArea*, phys_addr_t, uint32): undefined type 10000000!
allocate_commpage_entry(2, 24) -> 0x00000100
status_t arch_vm_set_memory_type(VMArea*, phys_addr_t, uint32): undefined type 10000000!
status_t arch_vm_set_memory_type(VMArea*, phys_addr_t, uint32): undefined type 10000000!
scheduler_init: found 1 logical cpu and 0 cache levels
scheduler switches: single core: true, cpu load tracking: false, core load tracking: false
scheduler: switching to low latency mode
slab memory manager: created area 0x81801000 (146)
allocate_commpage_entry(3, 8) -> 0x00000118
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module

module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module

module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module

module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
PCI: pci_module_init
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
usb xhci: failed to get pci x86 module
usb xhci: no devices found
module: Search for bus_managers/pci/x86/v1 failed.
usb uhci: failed to get pci x86 module
usb uhci: no devices found
module: Search for bus_managers/pci/x86/v1 failed.
usb ohci: failed to get pci x86 module
usb ohci: no devices found
module: Search for bus_managers/pci/x86/v1 failed.
usb ehci: failed to get pci x86 module
usb ehci: no devices found
usb error stack 0: no bus managers available
usb_disk: getting module failed: No such device
legacy_driver_add_preloaded: Failed to add "usb_disk": Device not accessible
get_boot_partitions(): boot volume message:
KMessage: buffer: 0x8240cd40 (size/capacity: 255/255), flags: 0xa
  field: "partition offset" (LLNG): 54520832 (0x33fec00)
  field: "packaged"   (BOOL): true
  field: "boot method"   (LONG): 0 (0x0)
  field: "disk identifier"   (RAWT): data at 0x8240cdf0, 79 bytes
get_boot_partitions(): boot method type: 0
intel: ep_std_ops(0x1)
intel: ep_std_ops(0x2)
intel: pm_std_ops(0x1)

```
intel: pm_std_ops(0x2)
dos_std_ops()
dos_std_ops()
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
  name: Generic
KDiskDeviceManager::InitialDeviceScan() returned error: No such file or directory
PANIC: did not find any boot partitions!
Welcome to Kernel Debugging Land...
Thread 14 "main2" running on CPU 0
frame            caller     <image>:function + offset
 0 801bc478 (+2145663880) 801b9734   <kernel_arm>  (nearest) + 0x00
initial commands:  syslog | tail 15
get_boot_partitions(): boot method type: 0
intel: ep_std_ops(0x1)
intel: ep_std_ops(0x2)
intel: pm_std_ops(0x1)
intel: pm_std_ops(0x2)
dos_std_ops()
dos_std_ops()
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
module: Search for bus_managers/pci/x86/v1 failed.
ahci: failed to get pci x86 module
  name: Generic
KDiskDeviceManager::InitialDeviceScan() returned error: No such file or directory
kdebug>
```