溪江朔

vilalge

Village Kernel开发指南

从零开始写内核

目录

[第一章: 序言 3](#_Toc1900653492)

[一、 Village内核特点： 3](#_Toc463990919)

[二、 Village内核目标： 3](#_Toc782253376)

[三、 进展说明 3](#_Toc437603498)

[四、 展望未来 4](#_Toc1817983558)

[第二章: 搭建开发环境 6](#_Toc452329790)

[一、 系统要求 6](#_Toc214670150)

[二、 搭建开发环境 6](#_Toc188684090)

[三、 克隆village-kernel项目 6](#_Toc1527637658)

[四、 使用vscode打开village-kernel项目 7](#_Toc1839118121)

[五、 创建rootfs文件系统镜像 7](#_Toc1326128376)

[六、 运行与调试代码 8](#_Toc1654326866)

[第三章: 内核结构 10](#_Toc800859153)

[一、 vk.application: 应用层代码 10](#_Toc1759768722)

[二、 vk.bootloader: 加载器代码 10](#_Toc1288124170)

[三、 vk.hardware: 硬件层代码 10](#_Toc720279783)

[四、 vk.kernel: 微内核代码 10](#_Toc376994742)

[五、 vk.library: C运行库代码 12](#_Toc1073870144)

[六、 vk.network: 网络层代码 12](#_Toc1082940820)

[七、 vk.scripts: 工具脚本 12](#_Toc1062453415)

[八、 vk.build：编译输出文件夹 12](#_Toc328021100)

[第四章: 执行流程 13](#_Toc460105851)

[第五章: 适配新平台 13](#_Toc2057908557)

[第六章: 编写驱动模块 13](#_Toc2044982564)

[第七章: 编写应用程序 13](#_Toc1693666560)

# 序言

## Village内核特点：

### 上层功能代码与底层驱动代码分离，可移植。

### 支持模块化，可裁剪，代码模块可分离

### 支持多线程，多任务

### 可动态加载模块，类似linux的insmod，rmmod

### 可运行应用程序，命令行run appname.exec

### 运行app时会根据编译时链接的动态库，进行加载so文件

## Village内核目标：

### 可在低端的嵌入式设备运行，也可以在高端的PC运行

### 让嵌入式开发者花费更少精力在底层，有更多精力搞好应用

### 适配更多通用设备，让开发者更快实现业务

### 不为项目更换MCU，需要重新适配底层而烦恼

## 进展说明

### 目前还处于开发阶段，各功能还不完善，代码还有点垃圾。

### 适配平台不多，目前只适配了cortex-m和i686平台。

### 已完成部分：

### 内存管理

### 中断管理

### 系统调度

### 任务管理

### 工作队列

### 线程同步（互斥锁，自旋锁，信号量）

### 文件系统（FAT）

### 动态加载（加载共享库，注册模块，运行程序）

### 正进行部分：

### 完善文件系统框架

### 完善注册模块逻辑

### 编写专用Bootloader

### 待完成部分：

### 编写GUI图形界面

### 适配POSIX接口

### 适配更多平台

### 其他文件系统

### 输入子系统

### 显示子系统

### 网络功能

## 展望未来

### 短时间内单靠一个人，搞不定太多，进度实在太慢。

### 编写适配平台底层代码，重写C库，重写文件系统，太耗时间。

### 寻找更多志同道合的码友，一起参与项目开发，人多力量大。

### 目前还很简陋，但星星之火，也可以燎原，一起把火烧起来。

# 搭建开发环境

## 系统要求

mac os / linux / windows（使用wsl子系统）

## 搭建开发环境

以mac os为例 (Linux一样可以ubuntu22.04测试过)

安装vscode, git

|  |
| --- |
| 安装简单，跳过。安装完成之后打开vscode，安装C/C++拓展插件，调试代码需要。 |

安装homebrew

|  |
| --- |
| /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)" |

安装交叉编译工具

|  |
| --- |
| brew install make gcc i686-elf-binutils i686-elf-gcc i386-elf-gdb |

安装qemu模拟器

|  |
| --- |
| brew install qemu |

如果出现too many open file错误时输入：

|  |
| --- |
| ulimit -n 4096 |

## 克隆village-kernel项目

ssh方式：

|  |
| --- |
| git clone git@github.com:village-kernel/village-kernel.git |

https方式：

|  |
| --- |
| git clone https://github.com/village-kernel/village-kernel.git |

## 使用vscode打开village-kernel项目

把项目目录village-kernel拉到vscode界面

接着打开vscode终端，拷贝配置文件

|  |
| --- |
| cp vk.scripts/configs/i686.config .config |

修改配置，进入Compiler选项

|  |
| --- |
| make menuconfig |

配置宿主机编译器：

|  |
| --- |
| () host compile prefix  (-13) host compile suffix |

配置交叉编译器：

|  |
| --- |
| (i686-elf-) cross compile prefix  () cross compile suffix |

编译项目

|  |
| --- |
| make |

## 创建rootfs文件系统镜像

**Mac OS**

切换到vscode终端，拷贝文件系统镜像

|  |
| --- |
| cp vk.scripts/rootfs.img rootfs.img |

右键选中rootfs.img，在Finder中打开，双击rootfs.img文件完成挂载

修改rootfs文件系统挂载路径

|  |
| --- |
| make menuconfig |

进入Compiler选项

|  |
| --- |
| (/Volumes/VILLAGE OS) rootfs path |

拷贝相关文件到文件系统

|  |
| --- |
| make rootfs |

**Linux**

切换到vscode终端，拷贝文件系统镜像

|  |
| --- |
| cp vk.scripts/rootfs.img rootfs.img |

终端挂载rootfs.img

|  |
| --- |
| sudo mount -o offset=512 rootfs.img /mnt |

修改rootfs文件系统挂载路径

|  |
| --- |
| make menuconfig |

进入Compiler选项

|  |
| --- |
| (/mnt) rootfs path |

拷贝相关文件到文件系统

|  |
| --- |
| sudo make rootfs |

## 运行与调试代码

切换到vscode debug界面

选择QEMU Debug x86\_64 kernel

# 内核结构

## vk.application: 应用层代码

## vk.bootloader: 加载器代码

## vk.hardware: 硬件层代码

arch：架构名称

mcu:

|  |
| --- |
| 存放mcu厂家提供的底层代码文件，包括启动代码，链接文件。  定义编译时的相关参数以及链接时的相关参数。 |

hal:

|  |
| --- |
| 存放与处理器相关的底层hal库代码，这里抽象化硬件资源以供driver调用。 |

driver:

|  |
| --- |
| 存放专用驱动代码 |

说明：

|  |
| --- |
| 适配新平台时，主要工作都在此目录 |

## vk.kernel: 微内核代码

**arch：**

|  |
| --- |
| 主要存放一些与CPU架构相关的代码，时钟、中断与调度。  启动代码之类的属于更加底层，存放在vk.hardware子模块。  这样做的目的是为了更好剥离底层代码，底层有底层的实现方式。  为了逻辑通用，切记不要在该目录编写太多不通用的底层的代码。 |
| arm：arm平台相关目录  x86：i686平台相关目录 |

**binutils：**

|  |
| --- |
| 主要存放与ELF文件相关的代码，例如Elf加载器、执行器，共享库工具，内核模块工具。 |

**bridge：**

|  |
| --- |
| 为了POSIX接口专门预留的目录，负责桥接内核层与用户层，目前还没有做适配。 |

**core：**

|  |
| --- |
| 与内核相关的核心代码，包括线程管理，内存管理，中断管理，模块管理，驱动管理等。  由于这里的代码不加入libkernel.so库，故增加了Environment类，用来导出相关接口。 |

**filesys：**

|  |
| --- |
| 文件系统相关目录，目前已规划好接入更多文件系统的框架，但还需继续打磨。  文件系统目前只适配了FAT，还不完善，后面会继续更改。 |

**sync：**

|  |
| --- |
| 这里是用来存放线程安全的相关目录，目前只简单实现了互斥锁，自旋锁，信号量。  poll，epoll，wait之类的需要安排时间来实现，但优先级会低一点。 |

**utilities：**

|  |
| --- |
| 这里是存放一些工具类的目录，比如解析器之类的，主要用来辅助处理文本文件。 |

## vk.library: C运行库代码

## vk.network: 网络层代码

## vk.scripts: 工具脚本

## vk.build：编译输出文件夹

# 执行流程

# 适配新平台

# 编写驱动模块

# 编写应用程序