溪江朔

vilalge

Village Kernel开发指南

从零开始写内核

目录

[第一章: 序言 4](#_Toc3207777)

[一、 Village内核特点： 4](#_Toc226016864)

[二、 Village内核目标： 4](#_Toc1914345352)

[三、 进展说明 4](#_Toc802331710)

[四、 说明 5](#_Toc739230457)

[第二章: 搭建开发环境 6](#_Toc1053392904)

[一、 系统要求 6](#_Toc519351660)

[二、 搭建开发环境 6](#_Toc1369808212)

[三、 克隆village-kernel项目 6](#_Toc1341923244)

[四、 使用vscode打开village-kernel项目 6](#_Toc830701114)

[五、 创建rootfs文件系统镜像 7](#_Toc802433851)

[六、 运行与调试代码 7](#_Toc308430597)

[第三章: 内核结构 9](#_Toc1915003568)

[一、 village-boot: 加载器代码 9](#_Toc1127549787)

[二、 village-kernel: 微内核代码 9](#_Toc1333568981)

[三、 village-os: 系统层代码 10](#_Toc7039928)

[四、 village-scripts: 工具脚本 11](#_Toc208469311)

[五、 village-docs：相关文档 11](#_Toc1197881720)

[六、 build：编译输出文件夹 11](#_Toc138877415)

[第四章: 执行流程 12](#_Toc1945473263)

[一、 启动加载内核并跳转执行 12](#_Toc2130605666)

[二、 设置中断向量表 16](#_Toc1947098384)

[三、 初始化段数据和构造函数 29](#_Toc1526726902)

[四、 跳转到main函数 31](#_Toc1564427558)

[五、 初始化内核模块 31](#_Toc1691677085)

[六、 初始化芯片时钟 33](#_Toc1480764962)

[七、 初始化中断向量 33](#_Toc28731251)

[八、 其他初始化略过 35](#_Toc1849798629)

[九、 初始化文件系统并挂载根目录 35](#_Toc444799984)

[十、 加载共享库及动态模块 36](#_Toc362755881)

[十一、 开始调度任务 37](#_Toc132018134)

[十二、 加载第一个应用程序 39](#_Toc478170787)

[十三、 应用层运行控制台 41](#_Toc732610035)

[第五章: 简易框架 42](#_Toc1453109994)

[一、 boot引导层 42](#_Toc1235635474)

[二、 app应用层 49](#_Toc1158545028)

[三、 debug调试 56](#_Toc432058247)

[第六章: 适配新平台 59](#_Toc960746822)

[四、 增加启动代码 59](#_Toc342295561)

[五、 增加框架平台 59](#_Toc2000287061)

[六、 增加初始化代码 59](#_Toc2115624089)

[七、 增加链接信息 60](#_Toc1407320444)

[八、 增加基本驱动 60](#_Toc449814250)

[九、 增加时钟管理 60](#_Toc885662310)

[十、 增加中断管理 60](#_Toc1117286813)

[十一、 增加调度管理 60](#_Toc642456723)

[十二、 修改或增加Kconfig 60](#_Toc222366345)

[十三、 修改或增加Makefile 60](#_Toc689614635)

[第七章: 编写驱动模块 60](#_Toc383927586)

[第八章: 编写功能模块 61](#_Toc1630062314)

[第九章: 编写应用程序 62](#_Toc1008426619)

# 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：

* 整理框架
* 优化代码

### 待完成部分：

* 完善GUI图形界面
* 适配更多平台
* 其他文件系统
* 网络功能

## 说明

* 目前还处于开发阶段，各功能还不完善，框架结构未确定，待优化。
* 适配平台不多，目前只适配了cortex-m和i686平台，其他平台待适配。

# 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.git |

https方式：

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

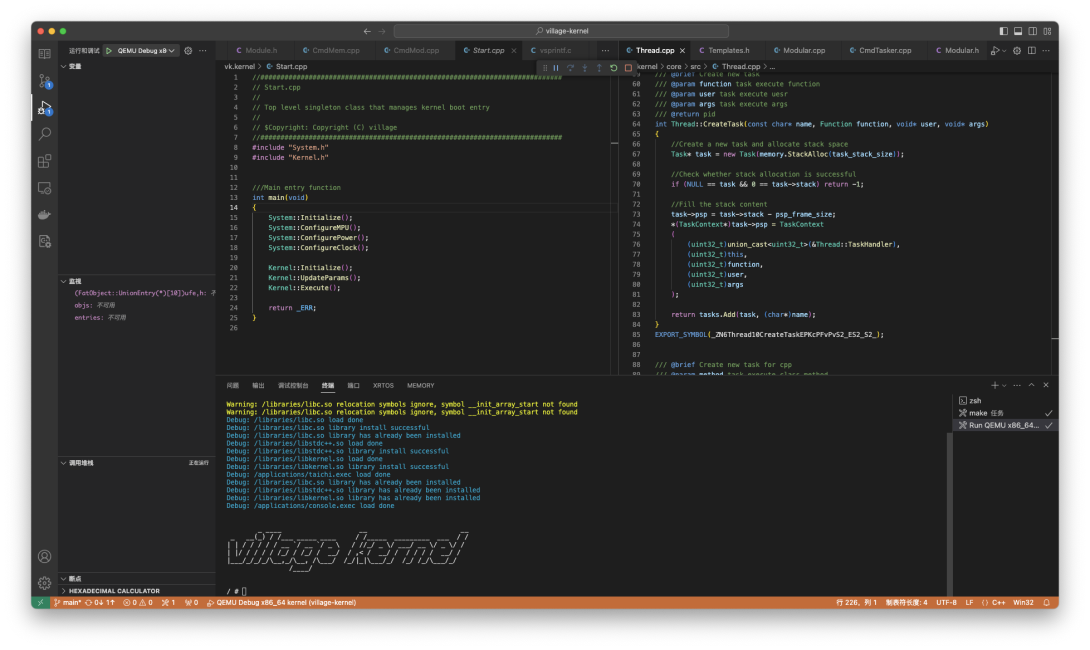
拷贝相关文件到文件系统

|  |
| --- |
| 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

## village-boot: 加载器代码

存放boot相关文件，目前有x86bios相关的启动代码。

## village-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 |

**drivers：**

|  |
| --- |
| 该目录用来存放驱动代码。 |

**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。 |

**kernel：**

|  |
| --- |
| 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，文件系统管理等。 |

**vklibs：**

|  |
| --- |
| 该目录是存放共享库代码，目前大部分会被生成为库的代码都该存放在该目录下面。 |

**libc：**

|  |
| --- |
| 内核的c库，归类到libc.so共享库。  目前只初步实现一些基础接口，例如stdio和string部分接口。 |

**libc++：**

|  |
| --- |
| 内核的stdc++库，归类到libstdc++.so共享库。  目前只重定义了new和delete。 |

**libm：**

|  |
| --- |
| 内核的数学库，归类到libm.so共享库。 |

libhw：

|  |
| --- |
| 这里用来存放与芯片厂商提供相关的代码，例如ST提供的HAL底层库之类。  该目录下的代码会归类到libhw.so共享库。  适配新平台时，主要工作都在此目录 |

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。 |

**libutils：**

|  |
| --- |
| 该目录下的代码会归类的libutils.so共享库。  **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，动态模块工具。  **fileutils：**  主要存放与文件操作相关的代码，例如FileStream, DirStream等。  **parser：**  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。  **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。 |

## village-os: 系统层代码

applications：

|  |
| --- |
| 存放操作系统相关的应用程序文件，console相关代码在该目录。 |

framework：

crt0：

|  |
| --- |
| 存放crt0启动文件和链接文件 |

vkgui：

|  |
| --- |
| 简易的GUI图形库 |

## village-scripts: 工具脚本

存放工具脚本，menuconfig和configs存放在该目录

## village-docs：相关文档

存放village的相关文档。

## build：编译输出文件夹

编译生成的临时文件存放目录。

# 执行流程

## 启动加载内核并跳转执行

启动代码也没那么复杂，只需要把存储在扇区1及之后的2879个扇区内的内核代码，读取到0x10000之后的sram空间里，然后跳转到该地址执行就行了。后续会由专用bootloader接管。

以x86为例：

bios启动模式下，以硬盘第一个扇区为启动扇区，结尾标志为0xaa55，将从这开始读取代码执行。目前只需要在这个扇区里面完成内核读取和跳转即可。

启动进入16位实模式->读取内核代码到指定sram位置->切换到32位保护模式->重新设置数据段和栈->跳转到内核。

这部分代码使用AT&T汇编代码编写，可以使用gcc编译。

汇编代码在village内核的占比非常少，能用C/C++写绝不使用汇编。

启动代码文件：village-boot/x86bios/boot.s

|  |
| --- |
| ############################################################################  # BootSection.s  # x86\_64 boot section, loading bootloader and switch to protected mode  #  # $Copyright: Copyright (C) village  ############################################################################  .org 0  .code16  .section ".text", "ax"  .set estack16, 0x9000  .set estack32, 0x2000000  .set appBaseAddr, 0x100000  .set appBaseSector, 1  .set appSectors, 2879  .global \_start  \_start:  movw %cs, %ax  movw %ax, %ds  movw %ax, %es  movw %ax, %ss  movw $estack16, %bp  movw %bp, %sp  call DisplayMsg  call ReadApplication  call SwitchToProtectedMode  jmp .  # Display boot message  DisplayMsg:  pusha  movw $0x0600, %ax # Clear screen  movw $0x0700, %bx # Page 0, white on black  movw $0x00, %cx # left: (0, 0)  movw $0x184f, %dx # right: (80, 50)  int $0x10 # Display interrupt  movw $0x0, %ax # Reset es  movw %ax, %es  movw $diskBootMsg, %ax # Set the display msg address  movw %ax, %bp  movw $0x1301, %ax # Display string  movw $0x0007, %bx # Page 0, Red on black  movw $26, %cx # String length  movw $0, %dx # Show in where, dh: row dl: col  int $0x10 # Display interrupt  popa  ret  diskBootMsg: .asciz "Booting from Hard Disk..."  # Loading application from disk  ReadApplication:  movw $appSectors, %cx  movl $appBaseAddr, %ebx  movl $appBaseSector, %esi  \_ReadAppData:  call ReadFromDisk  addl $1, %esi  addl $512, %ebx  loop \_ReadAppData  ret  # Read data from disk  ReadFromDisk:  pushl %ebx  pusha  movw $0x1f2, %dx # 0x1f2  movb $1, %al # read one sector  out %al, %dx  inc %dx # 0x1f3  movl %esi, %eax  out %al, %dx  inc %dx # 0x1f4  movb %ah, %al  out %al, %dx  inc %dx # 0x1f5  shrl $16, %eax  out %al, %dx  inc %dx # 0x1f6  movb $0xe0, %al # LBA28 mode  orb %ah, %al # LBA address 27 ~ 24  out %al, %dx  inc %dx # 0x1f7  movb $0x20, %al # read cmd  out %al, %dx  \_Wait1:  in %dx, %al  test $0x80, %al  jne \_Wait1  \_Wait2:  in %dx, %al  test $0x08, %al  je \_Wait2  movw $256, %cx  movw $0x1f0, %dx  \_Readw:  in %dx, %ax  movw %ax, (%ebx)  addl $2, %ebx  loop \_Readw  popa  popl %ebx  ret  # GDT start label  gdtStart:  # the GDT starts with a null 8-byte  .long 0x0 # 4 byte  .long 0x0 # 4 byte  # GDT for code segment. base = 0x00000000, length = 0xfffff for flags  gdtCode:  .word 0xffff # segment length, bits 0-15  .word 0x0 # segment base, bits 0-15  .byte 0x0 # segment base, bits 16-23  .byte 0x9a # 10011010b # flags (8 bits)  .byte 0xcf # 11001111b # flags (4 bits) + segment length, bits 16-19  .byte 0x0 # segment base, bits 24-31  # GDT for data segment. base and length identical to code segment some flags changed again  gdtData:  .word 0xffff  .word 0x0  .byte 0x0  .byte 0x92 # 10010010b  .byte 0xcf # 11001111b  .byte 0x0  # GDT end label  gdtEnd:  # GDT descriptor  gdtDescriptor:  .word gdtEnd - gdtStart - 1 # size (16 bit), always one less of its true size  .long gdtStart # address (32 bit)  # define some constants for later use  codeSeg = gdtCode - gdtStart  dataSeg = gdtData - gdtStart  # Switch to protected mode  SwitchToProtectedMode:  cli # disable interrupts  lgdt gdtDescriptor # load the GDT descriptor  mov %cr0, %eax  or $0x1, %eax # set 32-bit mode bit in cr0  mov %eax, %cr0  ljmp $codeSeg, $Setup # far jump by using a different segment  # Setup segment, stack and goto bootloader  .code32  Setup:  movw $dataSeg, %ax # update segment  movw %ax, %ds  movw %ax, %ss  movw %ax, %es  movw %ax, %fs  movw %ax, %gs    movl $estack32, %ebp # update stack  movl %ebp, %esp    jmp \*(appBaseAddr) # jmp to application  jmp .  # boot section end  bootSectionEnd:  .org 510  .word 0xaa55 # Magic word |

## 设置中断向量表

程序跳转到内核层，这时会进行中断向量表初始化，该中断向量表还不是最终的形态，内核初始化时会由Interrupt进行接管。

village-kernel/vklibs/libhw/x86/bios/cpu/crt0/crt0\_kernel.c

|  |
| --- |
| /// @brief IRQ\_Handler  void \_\_attribute\_\_ ((weak, naked)) IRQ\_Handler()  {  \_\_asm("jmp .");  }  /// @brief Stub\_Handler  void \_\_attribute\_\_ ((weak, naked)) Stub\_Handler()  {  \_\_asm("pusha");  \_\_asm("movw %ds, %ax");  \_\_asm("push %eax");  \_\_asm("movw $0x10, %ax");  \_\_asm("movw %ax, %ds");  \_\_asm("movw %ax, %es");  \_\_asm("movw %ax, %fs");  \_\_asm("movw %ax, %gs");    \_\_asm("call IRQ\_Handler");    \_\_asm("pop %eax");  \_\_asm("movw %ax, %ds");  \_\_asm("movw %ax, %es");  \_\_asm("movw %ax, %fs");  \_\_asm("movw %ax, %gs");  \_\_asm("popa");  \_\_asm("add $8, %esp ");  \_\_asm("sti");  \_\_asm("iret");  }  /// @brief Division\_By\_Zero\_Handler  void \_\_attribute\_\_ ((weak, naked)) Division\_By\_Zero\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $0");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Debug\_Handler  void \_\_attribute\_\_ ((weak, naked)) Debug\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $1");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Non\_Maskable\_Interrupt\_Handler  void \_\_attribute\_\_ ((weak, naked)) Non\_Maskable\_Interrupt\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $2");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Breakpoint\_Handler  void \_\_attribute\_\_ ((weak, naked)) Breakpoint\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $3");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Into\_Detected\_Overflow\_Handler  void \_\_attribute\_\_ ((weak, naked)) Into\_Detected\_Overflow\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $4");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Out\_Of\_Bounds\_Handler  void \_\_attribute\_\_ ((weak, naked)) Out\_Of\_Bounds\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $5");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Invalid\_Opcode\_Handler  void \_\_attribute\_\_ ((weak, naked)) Invalid\_Opcode\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $6");  \_\_asm("jmp Stub\_Handler");  }  /// @brief No\_Coprocessor\_Handler  void \_\_attribute\_\_ ((weak, naked)) No\_Coprocessor\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $7");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Doule\_Fault\_Handler  void \_\_attribute\_\_ ((weak, naked)) Doule\_Fault\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $8");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Coprocessor\_Segment\_Overrun\_Handler  void \_\_attribute\_\_ ((weak, naked)) Coprocessor\_Segment\_Overrun\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $9");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Bad\_TSS\_Handler  void \_\_attribute\_\_ ((weak, naked)) Bad\_TSS\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $10");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Segment\_Not\_Present\_Handler  void \_\_attribute\_\_ ((weak, naked)) Segment\_Not\_Present\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $11");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Stack\_Fault\_Handler  void \_\_attribute\_\_ ((weak, naked)) Stack\_Fault\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $12");  \_\_asm("jmp Stub\_Handler");  }  /// @brief General\_Protection\_Fault\_Handler  void \_\_attribute\_\_ ((weak, naked)) General\_Protection\_Fault\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $13");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Page\_Fault\_Handler  void \_\_attribute\_\_ ((weak, naked)) Page\_Fault\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $14");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Unknown\_Interrupt\_Handler  void \_\_attribute\_\_ ((weak, naked)) Unknown\_Interrupt\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $15");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Coprocessor\_Fault\_Handler  void \_\_attribute\_\_ ((weak, naked)) Coprocessor\_Fault\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $16");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Alignment\_Check\_Handler  void \_\_attribute\_\_ ((weak, naked)) Alignment\_Check\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $17");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Machine\_Check\_Handler  void \_\_attribute\_\_ ((weak, naked)) Machine\_Check\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $18");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Reserved\_IN\_19\_Handler  void \_\_attribute\_\_ ((weak, naked)) Reserved\_IN\_19\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $19");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Reserved\_IN\_20\_Handler  void \_\_attribute\_\_ ((weak, naked)) Reserved\_IN\_20\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $20");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Reserved\_IN\_21\_Handler  void \_\_attribute\_\_ ((weak, naked)) Reserved\_IN\_21\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $21");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Reserved\_IN\_22\_Handler  void \_\_attribute\_\_ ((weak, naked)) Reserved\_IN\_22\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $22");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Reserved\_IN\_23\_Handler  void \_\_attribute\_\_ ((weak, naked)) Reserved\_IN\_23\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $23");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Reserved\_IN\_24\_Handler  void \_\_attribute\_\_ ((weak, naked)) Reserved\_IN\_24\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $24");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Reserved\_IN\_25\_Handler  void \_\_attribute\_\_ ((weak, naked)) Reserved\_IN\_25\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $25");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Reserved\_IN\_26\_Handler  void \_\_attribute\_\_ ((weak, naked)) Reserved\_IN\_26\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $26");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Reserved\_IN\_27\_Handler  void \_\_attribute\_\_ ((weak, naked)) Reserved\_IN\_27\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $27");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Reserved\_IN\_28\_Handler  void \_\_attribute\_\_ ((weak, naked)) Reserved\_IN\_28\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $28");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Reserved\_IN\_29\_Handler  void \_\_attribute\_\_ ((weak, naked)) Reserved\_IN\_29\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $29");  \_\_asm("jmp Stub\_Handler");  }  /// @brief SVC\_Handler  void \_\_attribute\_\_ ((weak, naked)) SVC\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $30");  \_\_asm("jmp Stub\_Handler");  }  /// @brief PendSV\_Handler  void \_\_attribute\_\_ ((weak, naked)) PendSV\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $31");  \_\_asm("jmp Stub\_Handler");  }  /// @brief SysTick\_Handler  void \_\_attribute\_\_ ((weak, naked)) SysTick\_Handler()  {  \_\_asm("cli");  \_\_asm("push $0");  \_\_asm("push $32");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Keyboard\_Controller\_Handler  void \_\_attribute\_\_ ((weak, naked)) Keyboard\_Controller\_Handler()  {  \_\_asm("cli");  \_\_asm("push $1");  \_\_asm("push $33");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Reserved\_EX\_2\_Handler  void \_\_attribute\_\_ ((weak, naked)) Reserved\_EX\_2\_Handler()  {  \_\_asm("cli");  \_\_asm("push $2");  \_\_asm("push $34");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Serial\_Port\_COM2\_Handler  void \_\_attribute\_\_ ((weak, naked)) Serial\_Port\_COM2\_Handler()  {  \_\_asm("cli");  \_\_asm("push $3");  \_\_asm("push $35");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Serial\_Port\_COM1\_Handler  void \_\_attribute\_\_ ((weak, naked)) Serial\_Port\_COM1\_Handler()  {  \_\_asm("cli");  \_\_asm("push $4");  \_\_asm("push $36");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Line\_Print\_Terminal2\_Handler  void \_\_attribute\_\_ ((weak, naked)) Line\_Print\_Terminal2\_Handler()  {  \_\_asm("cli");  \_\_asm("push $5");  \_\_asm("push $37");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Floppy\_Controller\_Handler  void \_\_attribute\_\_ ((weak, naked)) Floppy\_Controller\_Handler()  {  \_\_asm("cli");  \_\_asm("push $6");  \_\_asm("push $38");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Line\_Print\_Terminal1\_Handler  void \_\_attribute\_\_ ((weak, naked)) Line\_Print\_Terminal1\_Handler()  {  \_\_asm("cli");  \_\_asm("push $7");  \_\_asm("push $39");  \_\_asm("jmp Stub\_Handler");  }  /// @brief RTC\_Timer\_Handler  void \_\_attribute\_\_ ((weak, naked)) RTC\_Timer\_Handler()  {  \_\_asm("cli");  \_\_asm("push $8");  \_\_asm("push $40");  \_\_asm("jmp Stub\_Handler");  }  /// @brief X86\_Assembly\_ACPI\_Handler  void \_\_attribute\_\_ ((weak, naked)) X86\_Assembly\_ACPI\_Handler()  {  \_\_asm("cli");  \_\_asm("push $9");  \_\_asm("push $41");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Reserved\_EX\_11\_Handler  void \_\_attribute\_\_ ((weak, naked)) Reserved\_EX\_11\_Handler()  {  \_\_asm("cli");  \_\_asm("push $10");  \_\_asm("push $42");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Reserved\_EX\_12\_Handler  void \_\_attribute\_\_ ((weak, naked)) Reserved\_EX\_12\_Handler()  {  \_\_asm("cli");  \_\_asm("push $11");  \_\_asm("push $43");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Mouse\_Controller\_Handler  void \_\_attribute\_\_ ((weak, naked)) Mouse\_Controller\_Handler()  {  \_\_asm("cli");  \_\_asm("push $12");  \_\_asm("push $44");  \_\_asm("jmp Stub\_Handler");  }  /// @brief Math\_Coprocessor\_Handler  void \_\_attribute\_\_ ((weak, naked)) Math\_Coprocessor\_Handler()  {  \_\_asm("cli");  \_\_asm("push $13");  \_\_asm("push $45");  \_\_asm("jmp Stub\_Handler");  }  /// @brief ATA\_Channel1\_Handler  void \_\_attribute\_\_ ((weak, naked)) ATA\_Channel1\_Handler()  {  \_\_asm("cli");  \_\_asm("push $14");  \_\_asm("push $46");  \_\_asm("jmp Stub\_Handler");  }  /// @brief ATA\_Channel2\_Handler  void \_\_attribute\_\_ ((weak, naked)) ATA\_Channel2\_Handler()  {  \_\_asm("cli");  \_\_asm("push $15");  \_\_asm("push $47");  \_\_asm("jmp Stub\_Handler");  }  /// @brief isr\_vector  void \* g\_pfnVectors[] \_\_attribute\_\_ ((section (".isr\_vector"), used)) =  {  &\_start,  &Division\_By\_Zero\_Handler,  &Debug\_Handler,  &Non\_Maskable\_Interrupt\_Handler,  &Breakpoint\_Handler,  &Into\_Detected\_Overflow\_Handler,  &Out\_Of\_Bounds\_Handler,  &Invalid\_Opcode\_Handler,  &No\_Coprocessor\_Handler,  &Doule\_Fault\_Handler,  &Coprocessor\_Segment\_Overrun\_Handler,  &Bad\_TSS\_Handler,  &Segment\_Not\_Present\_Handler,  &Stack\_Fault\_Handler,  &General\_Protection\_Fault\_Handler,  &Page\_Fault\_Handler,  &Unknown\_Interrupt\_Handler,  &Coprocessor\_Fault\_Handler,  &Alignment\_Check\_Handler,  &Machine\_Check\_Handler,  &Reserved\_IN\_19\_Handler,  &Reserved\_IN\_20\_Handler,  &Reserved\_IN\_21\_Handler,  &Reserved\_IN\_22\_Handler,  &Reserved\_IN\_23\_Handler,  &Reserved\_IN\_24\_Handler,  &Reserved\_IN\_25\_Handler,  &Reserved\_IN\_26\_Handler,  &Reserved\_IN\_27\_Handler,  &Reserved\_IN\_28\_Handler,  &Reserved\_IN\_29\_Handler,  &SVC\_Handler,  &PendSV\_Handler,  &SysTick\_Handler,  &Keyboard\_Controller\_Handler,  &Reserved\_EX\_2\_Handler,  &Serial\_Port\_COM2\_Handler,  &Serial\_Port\_COM1\_Handler,  &Line\_Print\_Terminal2\_Handler,  &Floppy\_Controller\_Handler,  &Line\_Print\_Terminal1\_Handler,  &RTC\_Timer\_Handler,  &X86\_Assembly\_ACPI\_Handler,  &Reserved\_EX\_11\_Handler,  &Reserved\_EX\_12\_Handler,  &Mouse\_Controller\_Handler,  &Math\_Coprocessor\_Handler,  &ATA\_Channel1\_Handler,  &ATA\_Channel2\_Handler,  }; |

## 初始化段数据和构造函数

跳转到内核之后：对数据段进行初始化->对构造函数进行初始化->跳转到main()。

其中对构造函数进行初始化时，也会把编译进内核的模块注册到对应的存储列表里。

数据初始化代码：village-kernel/vklibs/libhw/x86/bios/cpu/core/crt0\_kernel.c

|  |
| --- |
| //###########################################################################  // crt0\_kernel.c  // Low level file that manages kernel entry  //  // $Copyright: Copyright (C) village  //###########################################################################  /// @brief program entry main  /// @param argc  /// @param argv  /// @return  int main(int argc, char\* argv[]);  /// @brief \_start  /// @param argc  /// @param argv  void \_start(int argc, char\* argv[]);  /// @brief Initialize data and bss section  /// @param  void \_\_init\_data\_bss(void)  {  extern void \*\_sidata, \*\_sdata, \*\_edata;  extern void \*\_sbss, \*\_ebss;  void \*\*pSource, \*\*pDest;  //Copy data segment initializers from disk to SRAM  for (pSource = &\_sidata, pDest = &\_sdata; pDest != &\_edata; pSource++, pDest++)  \*pDest = \*pSource;  //Zero fill the bss segment.  for (pDest = &\_sbss; pDest != &\_ebss; pDest++)  \*pDest = 0;  }  /// @brief execute preinit\_arrary  /// @param  void \_\_preinit\_arrary(void)  {  extern void (\*\_\_preinit\_array\_start []) (void);  extern void (\*\_\_preinit\_array\_end []) (void);  int count = \_\_preinit\_array\_end - \_\_preinit\_array\_start;  for (int i = 0; i < count; i++)  \_\_preinit\_array\_start[i]();  }  /// @brief execute init\_arrary  /// @param  void \_\_init\_array(void)  {  extern void (\*\_\_init\_array\_start []) (void);  extern void (\*\_\_init\_array\_end []) (void);  int count = \_\_init\_array\_end - \_\_init\_array\_start;  for (int i = 0; i < count; i++)  \_\_init\_array\_start[i]();  }  /// @brief execute fini\_arrary  /// @param  void \_\_fini\_array(void)  {  extern void (\*\_\_fini\_array\_start []) (void);  extern void (\*\_\_fini\_array\_end []) (void);  int count = \_\_fini\_array\_end - \_\_fini\_array\_start;    for (int i = 0; i < count; i++)  {  \_\_fini\_array\_start[i]();  }  }  /// @brief \_start  /// @param  void \_start(int argc, char\* argv[])  {  \_\_init\_data\_bss();  \_\_preinit\_arrary();  \_\_init\_array();    main(argc, argv);    \_\_fini\_array();  for(;;) ;  } |

## 跳转到main函数

执行完数据初始化之后跳转到main函数。

主函数文件village-kernel/kernel/src/village.cpp

|  |
| --- |
| /// @brief Definition and export kernel  Kernel\* kernel = &Village::Instance();  EXPORT\_SYMBOL(kernel);  /// @brief Main entry function  /// @param argc  /// @param argv  /// @return  int main(int argc, char\* argv[])  {  kernel->Setup();  kernel->Start();  kernel->Exit();  return 0;  } |

## 初始化内核模块

Kernel负责初始化软件相关资源（在village-kernel/kernel目录）。

Kernel会初始化时钟，内存，中断，异常，线程，调度，设备，模块等。

village-kernel/kernel/src/Village.cpp

|  |
| --- |
| /// @brief Kernel Setup  void Village::Setup()  {  //Setup system  concreteSystem.Setup();  //Setup memory  concreteMemory.Setup();  //Setup debug  concreteDebug.Setup();    //Setup interrupt  concreteInterrupt.Setup();    //Setup scheduler  concreteScheduler.Setup();  //Setup thread  concreteThread.Setup();  //Setup work queue  concreteWorkQueue.Setup();  //Setup input event  concreteInputEvent.Setup();    //Setup symbol  concreteSymbol.Setup();    //Setup device  concreteDevice.Setup();  //Setup filesys  concreteFilesys.Setup();    //Setup feature  concreteFeature.Setup();  //Setup loader  concreteLoader.Setup();  } |

## 初始化芯片时钟

System负责初始化硬件相关资源（在village-kernel/arch目录）。

System会初始化时钟，供电等硬件模块。

village-kernel/arch/x86/bios/src/System.cpp

|  |
| --- |
| /// @brief System Setup  void ConcreteSystem::Setup()  {  //Set interrupt handler  kernel->interrupt.SetISR(IRQ\_Systick, (Method)&ConcreteSystem::SysTickHandler, this);  //Configure clock  ConfigureClock();  }  /// @brief Configure clock  void ConcreteSystem::ConfigureClock()  {  //Reset systicks  sysTicks = 0;  //Get the PIT value: hardware clock at 1193182 Hz  uint32\_t freq = 1000; //1000hz, 1ms  uint32\_t divider = 1193182 / freq;  uint8\_t low = low\_8(divider);  uint8\_t high = high\_8(divider);  //Send the command  PortByteOut(TIMER\_CMD, 0x36); //Command port  PortByteOut(TIMER\_CH0, low);  PortByteOut(TIMER\_CH0, high);  }  /// @brief System clock handler  void ConcreteSystem::SysTickHandler() { sysTicks++; } |

## 初始化中断向量

village-kernel/arch/x86/bios/src/ArchInterrupt.cpp

|  |
| --- |
| /// @brief ArchInterrupt Setup  void ArchInterrupt::Setup()  {  //Symbol defined in the linker script  extern void (\*\_svector [])(void);  extern void (\*\_evector [])(void);  //Calculate the size of isr vector  uint32\_t count = \_evector - \_svector;  //Set interrupt handler  for (uint32\_t i = 1; i < count; i++)  {  //The first func is \_start(), we don't need  SetIdtGate(i - 1, (uint32\_t)\_svector[i]);  }  //Remap the PIC  RemapPIC();  //Set IDT  SetIdt();  }  /// @brief Set idt gate  /// @param irq  /// @param handler  void ArchInterrupt::SetIdtGate(int irq, uint32\_t handler)  {  idt[irq].lowOffset = low\_16(handler);  idt[irq].highOffset = high\_16(handler);  idt[irq].sel = kernel\_code\_segment;  idt[irq].flags = 0x8E;  }  /// @brief Set idt  void ArchInterrupt::SetIdt()  {  idtReg.base = (uint32\_t)&idt;  idtReg.limit = idt\_entires \* sizeof(IdtGate) - 1;  \_\_asm volatile("lidtl (%0)" : : "r" (&idtReg));  }  /// @brief Remap the PIC  void ArchInterrupt::RemapPIC()  {  //Save masks  uint8\_t a1 = PortByteIn(PIC1\_DATA);  uint8\_t a2 = PortByteIn(PIC2\_DATA);  //starts the initialization sequence (in cascade mode)  PortByteOut(PIC1\_CMD, ICW1\_INIT | ICW1\_ICW4);  PortByteOut(PIC2\_CMD, ICW1\_INIT | ICW1\_ICW4);  //ICW2: Master PIC vector offset  PortByteOut(PIC1\_DATA, 0x20);  //ICW2: Slave PIC vector offset  PortByteOut(PIC2\_DATA, 0x28);    //ICW3: tell Master PIC that there is a slave PIC at IRQ2 (0000 0100)  PortByteOut(PIC1\_DATA, 0x04);  //ICW3: tell Slave PIC its cascade identity (0000 0010)  PortByteOut(PIC2\_DATA, 0x02);  //ICW4: have the PICs use 8086 mode (and not 8080 mode)  PortByteOut(PIC1\_DATA, ICW4\_8086);  PortByteOut(PIC2\_DATA, ICW4\_8086);  //Restore saved masks  PortByteOut(PIC1\_DATA, a1);  PortByteOut(PIC2\_DATA, a2);  } |

## 其他初始化略过

## 初始化文件系统并挂载根目录

village-kernel/kernel/src/FileSystem.cpp

|  |
| --- |
| /// @brief File system setup  void ConcreteFileSystem::Setup()  {  if (!InitDisk()) return;  if (!ReadMBR()) return;  InitVolumes();  MountSystemNode();  }  /// @brief Init volumes  void ConcreteFileSystem::InitVolumes()  {  for (uint8\_t i = 0; i < 4; i++)  {  FileSys\* fs = fileSys.GetItem(mbr->dpt[i].systemID);  if (NULL != fs)  {  FileVol\* volume = fs->CreateVolume();  if (volume->Setup(&diskdrv, mbr->dpt[i].relativeSectors))  {  AttachVolume(volume);  }  else delete volume;  }  }  }  /// @brief Mount node  void ConcreteFileSystem::MountSystemNode()  {  //Mount root node "/"  for (volumes.Begin(); !volumes.IsEnd(); volumes.Next())  {  char\* volumelab = volumes.GetName();  if (0 == strcmp(volumelab, "/media/VILLAGE OS"))  {  mounts.Add(new MountNode((char\*)"/", volumelab, 0755));  return;  }  }  kernel->debug.Output(Debug::\_Lv2, "Mount system node failed, '/media/VILLAGE OS' not found");  } |

## 加载共享库及动态模块

加载共享库和模块。

village-kernel/kernel/src/Loader.cpp

|  |
| --- |
| /// @brief Loader setup  void ConcreteLoader::Setup()  {  //Loading libraries  Loading(\_Load\_Lib, "/libraries/\_load\_.rc");  //Loading modules  Loading(\_Load\_Mod, "/modules/\_load\_.rc");  }  /// @brief Loader load  /// @param filename rc file path  void ConcreteLoader::Loading(int type, const char\* filename)  {  RcParser\* parser = new RcParser(filename);  List<char\*>& runcmds = parser->GetRunCmds();  for (runcmds.End(); !runcmds.IsBegin(); runcmds.Prev())  {  if (\_Load\_Lib == type)  {  if (!libraryTool.Install(runcmds.Item())) break;  }  else if (\_Load\_Mod == type)  {  if (!moduleTool.Install(runcmds.Item())) break;  }  }  parser->Release();  delete parser;  } |

## 开始调度任务

模块任务注册完成之后则开始进行任务调度了，滴答定时器触发任务调度。 SysTickHandler->PendSVHandler，保存当前任务栈及sp现场，切换下个任务，还原下个任务栈及sp。

village-kernel/arch/x86/bios/src/Scheduler.cpp

|  |
| --- |
| /// @brief Start scheduler  void ConcreteScheduler::Start()  {  //Clear start schedule flag  isStartSchedule = false;  //Get frist task psp  uint32\_t psp = kernel->thread.GetTaskPSP();  //Set frist task esp  \_\_asm volatile("movl %0, %%esp" : "=r"(psp));  //Set start schedule flag  isStartSchedule = true;  //Set interrupt flag  \_\_asm volatile("sti");  //Execute thread idle task  kernel->thread.IdleTask();  }  /// @brief Rescheduler task  /// @param access scheduler access  void ConcreteScheduler::Sched(ConcreteScheduler::Access access)  {  if (false == isStartSchedule) return;  //Trigger PendSV directly  \_\_asm volatile("int $31");  }  /// @brief PendSV handler  void \_\_attribute\_\_((naked)) ConcreteScheduler::PendSVHandler()  {  uint32\_t psp = 0;  //Push old task registers  \_\_asm volatile("pushl %ebp");  \_\_asm volatile("pushl %ebx");  \_\_asm volatile("pushl %esi");  \_\_asm volatile("pushl %edi");  \_\_asm volatile("movl %%esp, %0" : "=r"(psp));  //Save old task psp  kernel->thread.SaveTaskPSP(psp);  //Select next task  kernel->thread.SelectNextTask();  //Get new task psp  psp = kernel->thread.GetTaskPSP();  //Set new task esp  \_\_asm volatile("movl %0, %%esp" : "=r"(psp));  //Pop new task registers  \_\_asm volatile("popl %edi");  \_\_asm volatile("popl %esi");  \_\_asm volatile("popl %ebx");  \_\_asm volatile("popl %ebp");  \_\_asm volatile("sti");  \_\_asm volatile("ret");  } |

## 加载第一个应用程序

在初始化Loader时则会创建一个运行“/applications/taichi.exec”的任务，系统调度末尾是该任务。

village-kernel/kernel/src/Loader.cpp

|  |
| --- |
| /// @brief Loader execute  void ConcreteLoader::Execute()  {  //Execute the first application of the village  executor.Run(ElfExecutor::\_Background, "/applications/taichi.exec");  } |

village-kernel/vklibs/libutils/binutils/src/B aseExecutor.cpp

|  |
| --- |
| /// @brief BaseExecutor Initialize  /// @param args run args  /// @return pid  int BaseExecutor::Run(Behavior behavior, const char\* args)  {  //Split args  regex.Split(args);  //Set argc and argv  int argc = regex.Size();  char\*\* argv = regex.ToArray();  //Run with argc and argv  return Run(behavior, argv[0], argc, argv);  }  /// @brief BaseExecutor Initialize  /// @param path file path  /// @param argc running argc  /// @param argv running argv  /// @return pid  int BaseExecutor::Run(Behavior behavior, const char\* path, int argc, char\* argv[])  {  //Set argc and argv  this->argc = argc;  this->argv = argv;  //Load, parser file and create task  if ((pid = Execute(path)) == 0) return -1;  //Wait for task done  if (behavior == \_Foreground) kernel->thread.WaitForTask(pid);  return pid;  } |

village-kernel/vklibs/libutils/binutils/src/ElfExecutor.cpp

|  |
| --- |
| /// @brief ElfExecutor Execute  /// @param path  /// @return pid  int ElfExecutor::Execute(const char\* path)  {  //Load, parser and execute bin file  if (!elf.Load(path)) return 0;  //Create a sandboxed thread to run the app  return kernel->thread.CreateTask(path, (Method)&ElfExecutor::Sandbox, this);  }  /// @brief ElfExecutor execute app  void ElfExecutor::Sandbox()  {  elf.Execute(NULL, argc, argv);  elf.Exit();  } |

关于解析并加载elf文件，其实也不是很复杂，把elf通过文件系统读取出来，再按照数据读取内容到sram，并重定位相关entry，并执行初始化则可以。难点在加载内容到sram和重定位entry，重点重定位。相关代码在village-kernel/vklibs/libutils/binutils/src/ElfLoader.cpp，代码太多这里不粘贴了。

还有elf文件类型也得是DYN (Position-Independent Executable file)，如何生成该类型文件与ld flags相关，以下ld flags可以参考。

village-os/framework/ctr0/Makefile

|  |
| --- |
| #######################################  # compiler flags  #######################################  # application ld flags  APPLDFLAGS += $(MCU) $(LDSCRIPT-APP)  APPLDFLAGS += -ffreestanding -nostdlib  APPLDFLAGS += -Wl,-Map=$(APPS\_DIR)/$(name).map,--cref  APPLDFLAGS += -Wl,--gc-sections  APPLDFLAGS += -Wl,--no-warn-rwx-segment  APPLDFLAGS += -Wl,--unresolved-symbols=ignore-in-shared-libs  APPLDFLAGS += -pie  ifeq ($(CONFIG\_X86), y)  APPLDFLAGS += -Wl,-m,elf\_i386  endif |

## 应用层运行控制台

内核调用并执行taichi.exec之后，则来到了应用层，在taichi应用程序里面后台执行了console程序。

village-os/applications/taichi/src/Taichi.cpp

|  |
| --- |
| //###########################################################################  // Taichi.cpp  // The overall framework of the taichi  //  // $Copyright: Copyright (C) village  //###########################################################################  #include "Kernel.h"  #include "Taichi.h"  #include "ElfExecutor.h"  /// @brief Constructor  Taichi::Taichi()  {  }  /// @brief Destructor  Taichi::~Taichi()  {  }  /// @brief Initialize  void Taichi::Initialize()  {  }  /// @brief Execute  void Taichi::Execute()  {  ElfExecutor\* console = new ElfExecutor();  console->Run(ElfExecutor::\_Background, "/applications/console.exec serial0");  while (1) {}  }  /// @brief main  int main(void)  {  Taichi taichi;  taichi.Initialize();  taichi.Execute();  return 0;  } |

# 简易框架

## boot引导层

 启动加载并跳转到app

启动代码也没那么复杂，只需要把存储在扇区1及之后的2879个扇区内的app代码，读取到0x10000之后的sram空间里，然后跳转到该地址执行就行了。

以x86为例：

bios启动模式下，以硬盘第一个扇区为启动扇区，结尾标志为0xaa55，将从这开始读取代码执行。目前只需要在这个扇区里面完成内核读取和跳转即可。

执行过程：

启动进入16位实模式->读取内核代码到指定sram位置->设置GDT->切换到32位保护模式->重新设置数据段和栈->跳转到app。

说明：

这部分代码使用AT&T汇编代码编写，可以使用gcc编译。

启动代码文件：01\_boot/boot/boot.s

|  |
| --- |
| ############################################################################  # BootSection.s  # x86\_64 boot section, loading bootloader and switch to protected mode  #  # $Copyright: Copyright (C) village  ############################################################################  .org 0  .code16  .section ".text", "ax"  .set estack16, 0x9000  .set estack32, 0x2000000  .set appBaseAddr, 0x100000  .set appBaseSector, 1  .set appSectors, 2879  .global \_start  \_start:  movw %cs, %ax  movw %ax, %ds  movw %ax, %es  movw %ax, %ss  movw $estack16, %bp  movw %bp, %sp  call DisplayMsg  call ReadApplication  call SwitchToProtectedMode  jmp .  # Display boot message  DisplayMsg:  pusha  movw $0x0600, %ax # Clear screen  movw $0x0700, %bx # Page 0, white on black  movw $0x00, %cx # left: (0, 0)  movw $0x184f, %dx # right: (80, 50)  int $0x10 # Display interrupt  movw $0x0, %ax # Reset es  movw %ax, %es  movw $diskBootMsg, %ax # Set the display msg address  movw %ax, %bp  movw $0x1301, %ax # Display string  movw $0x0007, %bx # Page 0, Red on black  movw $26, %cx # String length  movw $0, %dx # Show in where, dh: row dl: col  int $0x10 # Display interrupt  popa  ret  diskBootMsg: .asciz "Booting from Hard Disk..."  # Loading application from disk  ReadApplication:  movw $appSectors, %cx  movl $appBaseAddr, %ebx  movl $appBaseSector, %esi  \_ReadAppData:  call ReadFromDisk  addl $1, %esi  addl $512, %ebx  loop \_ReadAppData  ret  # Read data from disk  ReadFromDisk:  pushl %ebx  pusha  movw $0x1f2, %dx # 0x1f2  movb $1, %al # read one sector  out %al, %dx  inc %dx # 0x1f3  movl %esi, %eax  out %al, %dx  inc %dx # 0x1f4  movb %ah, %al  out %al, %dx  inc %dx # 0x1f5  shrl $16, %eax  out %al, %dx  inc %dx # 0x1f6  movb $0xe0, %al # LBA28 mode  orb %ah, %al # LBA address 27 ~ 24  out %al, %dx  inc %dx # 0x1f7  movb $0x20, %al # read cmd  out %al, %dx  \_Wait1:  in %dx, %al  test $0x80, %al  jne \_Wait1  \_Wait2:  in %dx, %al  test $0x08, %al  je \_Wait2  movw $256, %cx  movw $0x1f0, %dx  \_Readw:  in %dx, %ax  movw %ax, (%ebx)  addl $2, %ebx  loop \_Readw  popa  popl %ebx  ret  # GDT start label  gdtStart:  # the GDT starts with a null 8-byte  .long 0x0 # 4 byte  .long 0x0 # 4 byte  # GDT for code segment. base = 0x00000000, length = 0xfffff for flags  gdtCode:  .word 0xffff # segment length, bits 0-15  .word 0x0 # segment base, bits 0-15  .byte 0x0 # segment base, bits 16-23  .byte 0x9a # 10011010b # flags (8 bits)  .byte 0xcf # 11001111b # flags (4 bits) + segment length, bits 16-19  .byte 0x0 # segment base, bits 24-31  # GDT for data segment. base and length identical to code segment some flags changed again  gdtData:  .word 0xffff  .word 0x0  .byte 0x0  .byte 0x92 # 10010010b  .byte 0xcf # 11001111b  .byte 0x0  # GDT end label  gdtEnd:  # GDT descriptor  gdtDescriptor:  .word gdtEnd - gdtStart - 1 # size (16 bit), always one less of its true size  .long gdtStart # address (32 bit)  # define some constants for later use  codeSeg = gdtCode - gdtStart  dataSeg = gdtData - gdtStart  # Switch to protected mode  SwitchToProtectedMode:  cli # disable interrupts  lgdt gdtDescriptor # load the GDT descriptor  mov %cr0, %eax  or $0x1, %eax # set 32-bit mode bit in cr0  mov %eax, %cr0  ljmp $codeSeg, $Setup # far jump by using a different segment  # Setup segment, stack and goto bootloader  .code32  Setup:  movw $dataSeg, %ax # update segment  movw %ax, %ds  movw %ax, %ss  movw %ax, %es  movw %ax, %fs  movw %ax, %gs    movl $estack32, %ebp # update stack  movl %ebp, %esp    jmp \*(appBaseAddr) # jmp to application  jmp .  # boot section end  bootSectionEnd:  .org 510  .word 0xaa55 # Magic word |

01\_boot/boot/boot.lds

|  |
| --- |
| OUTPUT\_FORMAT("elf32-i386", "elf32-i386","elf32-i386")  OUTPUT\_ARCH(i386)  ENTRY(\_start)  MEMORY  {  RAM (xrw) : ORIGIN = 0x7c00, LENGTH = 512  }  SECTIONS  {  .text :  {  . = ALIGN(8);  \*(.text)  \*(.text\*)  . = ALIGN(8);  } > RAM  .rodata :  {  . = ALIGN(8);  \*(.rodata)  \*(.rodata\*)  . = ALIGN(8);  } > RAM  .data :  {  . = ALIGN(8);  \*(.data)  \*(.data\*)  . = ALIGN(8);  } > RAM  .bss :  {  . = ALIGN(8);  \*(.bss)  \*(.bss\*)  . = ALIGN(8);  } > RAM  } |

01\_boot/boot/Makefile

|  |
| --- |
| ###########################################################################  # Makefile  # The Makefile of x86bios boot  #  # $Copyright: Copyright (C) village  ############################################################################  ######################################  # ASFLAGS  ######################################  ASFLAGS += -g -gdwarf-2 -DDEBUG  ######################################  # link script  ######################################  LDSCRIPT-BOOT := -T boot.lds  #######################################  # compiler flags  #######################################  # boot loader ld flags  LDFLAGS += $(LDSCRIPT-BOOT) -ffreestanding -nostdlib  LDFLAGS += -Wl,--no-warn-rwx-segment  LDFLAGS += -Wl,-m,elf\_i386  #######################################  # build task  #######################################  all:  i686-elf-gcc -x assembler-with-cpp -c $(ASFLAGS) boot.s -o boot.o  i686-elf-gcc $(LDFLAGS) boot.o -o village-boot.elf  i686-elf-objcopy -O binary -S village-boot.elf village-boot.bin  clean:  rm \*.o \*.elf \*.bin |

## app应用层

在boot跳转到app时，是直接跳转到app的基地址所指向的地址（ jmp \*(appBaseAddr)  ），因此需要保证第一个字节是程序的入口地址。在链接文件中，我定义了isr\_vector扇区（用来存储中断向量表，这里还没中断相关内容），isr\_vector在链接文件中处于最开始位置，能保证第一位置就是程序的入口，因此这里使用了isr\_vector扇区来存储程序的入口位置。

其实kernel本质上是一个功能更加复杂的app。

01\_boot/app/crt0.o

|  |
| --- |
| //###########################################################################  // crt0.c  // Low level file that manages kernel entry  //  // $Copyright: Copyright (C) village  //###########################################################################  /// @brief program entry main  /// @param argc  /// @param argv  /// @return  int main(int argc, char\* argv[]);  /// @brief \_start  /// @param argc  /// @param argv  void \_start(int argc, char\* argv[]);  /// @brief isr\_vector  void \* g\_pfnVectors[] \_\_attribute\_\_ ((section (".isr\_vector"), used)) =  {  &\_start,  };  /// @brief Initialize data and bss section  /// @param  void \_\_init\_data\_bss(void)  {  extern void \*\_sidata, \*\_sdata, \*\_edata;  extern void \*\_sbss, \*\_ebss;  void \*\*pSource, \*\*pDest;  //Copy data segment initializers from disk to SRAM  for (pSource = &\_sidata, pDest = &\_sdata; pDest != &\_edata; pSource++, pDest++)  \*pDest = \*pSource;  //Zero fill the bss segment.  for (pDest = &\_sbss; pDest != &\_ebss; pDest++)  \*pDest = 0;  }  /// @brief execute preinit\_arrary  /// @param  void \_\_preinit\_arrary(void)  {  extern void (\*\_\_preinit\_array\_start []) (void);  extern void (\*\_\_preinit\_array\_end []) (void);  int count = \_\_preinit\_array\_end - \_\_preinit\_array\_start;  for (int i = 0; i < count; i++)  \_\_preinit\_array\_start[i]();  }  /// @brief execute init\_arrary  /// @param  void \_\_init\_array(void)  {  extern void (\*\_\_init\_array\_start []) (void);  extern void (\*\_\_init\_array\_end []) (void);  int count = \_\_init\_array\_end - \_\_init\_array\_start;  for (int i = 0; i < count; i++)  \_\_init\_array\_start[i]();  }  /// @brief execute fini\_arrary  /// @param  void \_\_fini\_array(void)  {  extern void (\*\_\_fini\_array\_start []) (void);  extern void (\*\_\_fini\_array\_end []) (void);  int count = \_\_fini\_array\_end - \_\_fini\_array\_start;    for (int i = 0; i < count; i++)  {  \_\_fini\_array\_start[i]();  }  }  /// @brief \_start  /// @param  void \_start(int argc, char\* argv[])  {  \_\_init\_data\_bss();  \_\_preinit\_arrary();  \_\_init\_array();    main(argc, argv);    \_\_fini\_array();  for(;;) ;  } |

01\_boot/app/main.c

|  |
| --- |
| /// @brief print  /// @param string  void print(char\* string)  {  char\* videoMemory = (char\*)0xb8000;  for (int i = 0; 0 != string[i]; i++)  {  \*videoMemory = string[i];  videoMemory = videoMemory + 2;  }  }  /// @brief main  /// @return  int main()  {  print("Village-Kernel, Hello C world!");  while (1) {}  } |

01\_boot/app/app.lds

|  |
| --- |
| OUTPUT\_FORMAT("elf32-i386", "elf32-i386","elf32-i386")  OUTPUT\_ARCH(i386)  ENTRY(\_start)  \_estack = 0x2000000;  \_Min\_Heap\_Size = 0x400;  \_Min\_Stack\_Size = 0x800;  MEMORY  {  RAM (xrw) : ORIGIN = 0x100000, LENGTH = 10M  }  SECTIONS  {  \_sivector = LOADADDR(.isr\_vector);  .isr\_vector :  {  . = ALIGN(4);  \_svector = .;  KEEP(\*(.isr\_vector))  . = ALIGN(4);  \_evector = .;  } > RAM  .text :  {  . = ALIGN(4);  \*(.text)  \*(.text\*)  KEEP (\*(.init))  KEEP (\*(.fini))  . = ALIGN(4);  \_etext = .;  } > RAM  .rodata :  {  . = ALIGN(4);  \*(.rodata)  \*(.rodata\*)  . = ALIGN(4);  } > RAM  .x86.extab : { \*(.gcc\_except\_table.\* .got.plt ) } > RAM  .x86 : {  \_\_exidx\_start = .;  \*(.x86.extab\*)  \_\_exidx\_end = .;  } > RAM  .preinit\_array :  {  PROVIDE\_HIDDEN (\_\_preinit\_array\_start = .);  KEEP (\*(SORT(.preinit\_array.\*)))  KEEP (\*(SORT(.preinit\_array\*)))  PROVIDE\_HIDDEN (\_\_preinit\_array\_end = .);  } > RAM  .init\_array :  {  PROVIDE\_HIDDEN (\_\_init\_array\_start = .);  KEEP (\*(SORT(.init\_array.\*)))  KEEP (\*(SORT(.init\_array\*)))  KEEP (\*(SORT(.ctors\*)))  PROVIDE\_HIDDEN (\_\_init\_array\_end = .);  } > RAM  .fini\_array :  {  PROVIDE\_HIDDEN (\_\_fini\_array\_start = .);  KEEP (\*(SORT(.fini\_array.\*)))  KEEP (\*(SORT(.fini\_array\*)))  KEEP (\*(SORT(.dtors\*)))  PROVIDE\_HIDDEN (\_\_fini\_array\_end = .);  } > RAM  \_sidata = LOADADDR(.data);  .data :  {  . = ALIGN(4);  \_sdata = .;  \*(.data)  \*(.data\*)  . = ALIGN(4);  \_edata = .;  } > RAM  .bss :  {  . = ALIGN(4);  \_sbss = .;  \_\_bss\_start\_\_ = \_sbss;  \*(.bss)  \*(.bss\*)  \*(COMMON)  . = ALIGN(4);  \_ebss = .;  \_\_bss\_end\_\_ = \_ebss;  } > RAM  .\_user\_heap\_stack :  {  . = ALIGN(4);  PROVIDE ( end = . );  PROVIDE ( \_end = . );  . = . + \_Min\_Heap\_Size;  . = . + \_Min\_Stack\_Size;  . = ALIGN(4);  } > RAM  /\*/DISCARD/ :  {  libc.a ( \* )  libm.a ( \* )  libgcc.a ( \* )  }\*/  } |

01\_boot/app/Makefile

|  |
| --- |
| ###########################################################################  # Makefile  # The Makefile of app  #  # $Copyright: Copyright (C) village  ############################################################################  ######################################  # CFLAGS  ######################################  CFLAGS += -g -gdwarf-2 -DDEBUG  CFLAGS += -Wall -fdata-sections -ffunction-sections -fno-common  ######################################  # link script  ######################################  LDSCRIPT-BOOT := -T app.lds  #######################################  # compiler flags  #######################################  # app ld flags  LDFLAGS += $(LDSCRIPT-BOOT) -ffreestanding -nostdlib  LDFLAGS += -Wl,--gc-sections  LDFLAGS += -Wl,--no-warn-rwx-segment  LDFLAGS += -Wl,-m,elf\_i386  LDFLAGS += -Wl,-static -pie  #######################################  # build task  #######################################  all:  i686-elf-gcc -c $(CFLAGS) crt0.c -o crt0.o  i686-elf-gcc -c $(CFLAGS) main.c -o main.o  i686-elf-gcc $(LDFLAGS) crt0.o main.o -o village-kernel.elf  i686-elf-objcopy -O binary -S village-kernel.elf village-kernel.bin  clean:  rm \*.o \*.elf \*.bin |

## debug调试

要通过vscode进行debug，还需要以下设置。

1. 增加Makefile，把village-boot.bin和village-kernel.bin合并在一起。
2. 增加.vscode/launch.json文件，增加debug项目。
3. 增加.vscode/tasks.json文件，配置调试相关条件。

01\_boot/Makefile

|  |
| --- |
| all:  cd boot && make && cd ..  cd app && make && cd ..  dd if=/dev/zero of=village-os.img bs=512 count=2880  dd if=boot/village-boot.bin of=village-os.img bs=512 seek=0 conv=notrunc  dd if=app/village-kernel.bin of=village-os.img bs=512 seek=1 conv=notrunc  clean:  cd boot && make clean && cd ..  cd app && make clean && cd .. |

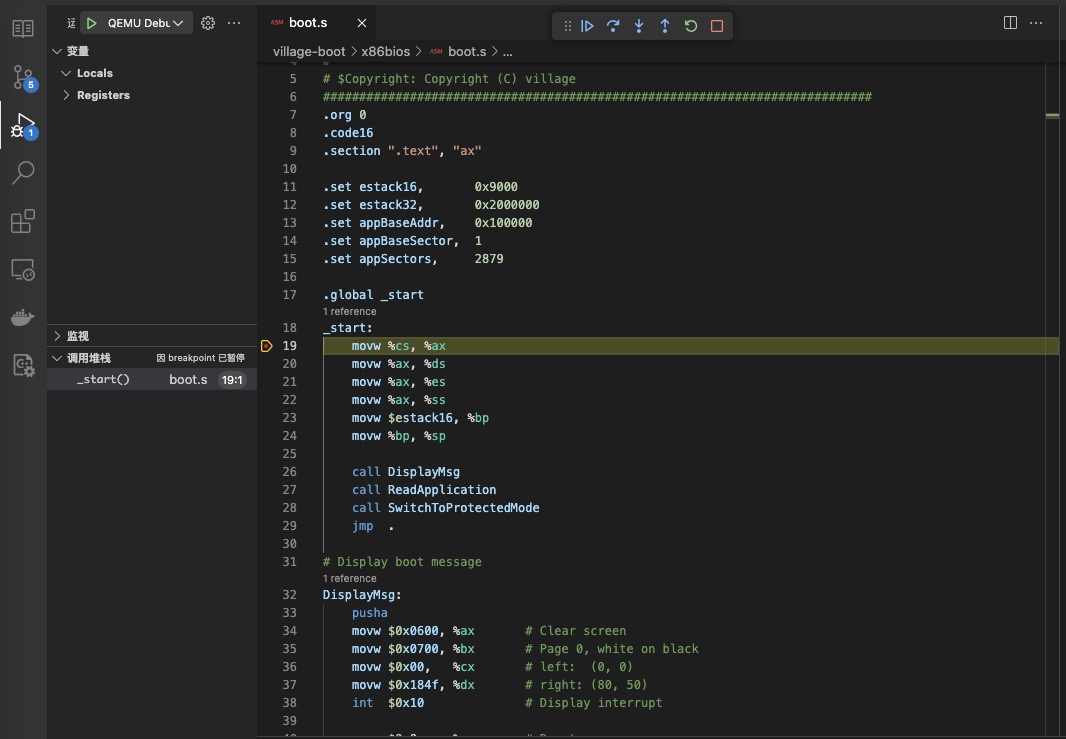
01\_boot/.vscode/launch.json

|  |
| --- |
| {  // 使用 IntelliSense 了解相关属性。  // 悬停以查看现有属性的描述。  // 欲了解更多信息，请访问: https://go.microsoft.com/fwlink/?linkid=830387  "version": "0.2.0",  "configurations": [  {  "name": "QEMU Debug x86 bios boot",  "type": "cppdbg",  "request": "launch",  "program": "${workspaceFolder}/boot/village-boot.elf",  "cwd": "${workspaceFolder}",  "miDebuggerPath": "i386-elf-gdb",  "miDebuggerServerAddress": "localhost:1234",  "stopAtEntry": true,  "preLaunchTask": "Run QEMU x86 bios"  },  {  "name": "QEMU Debug x86 bios kernel",  "type": "cppdbg",  "request": "launch",  "program": "${workspaceFolder}/app/village-kernel.elf",  "cwd": "${workspaceFolder}",  "miDebuggerPath": "i386-elf-gdb",  "miDebuggerServerAddress": "localhost:1234",  "stopAtEntry": true,  "preLaunchTask": "Run QEMU x86 bios"  }  ]  } |

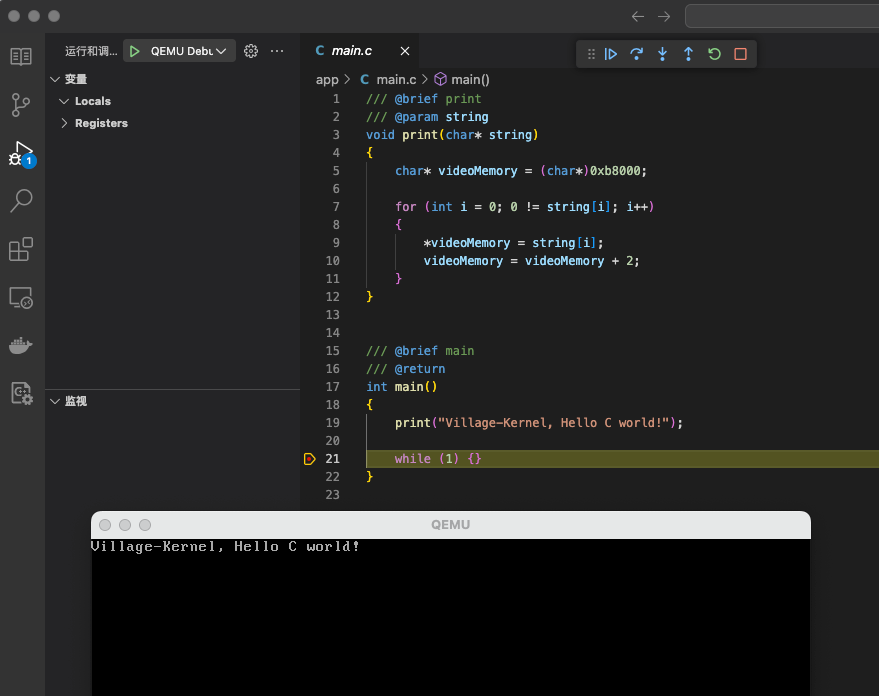
01\_boot/.vscode/tasks.json

|  |
| --- |
| {  "version": "2.0.0",  "tasks": [  {  "type":"shell",  "label": "Build",  "command": "make",  "args": [  "all"  ],  "detail": "Build project"  },  {  "label": "Run QEMU x86 bios",  "type":"shell",  "isBackground":true,  "dependsOn": ["Build"],  //"command": "qemu-system-i386 -hda ${workspaceFolder}/village-os.img -monitor null -serial stdio -s -S -nographic",  "command": "qemu-system-i386 -hda ${workspaceFolder}/village-os.img -monitor null -serial stdio -s -S",  "presentation": {  "echo": true,  "reveal": "always",  "focus": true,  "panel": "dedicated",  "showReuseMessage": true,  "clear": true,  },  "problemMatcher":  {  "owner": "external",  "pattern": [  {  "regexp": ".",  "file": 1,  "location": 2,  "message": 3  }  ],  "background": {  "activeOnStart": true,  "beginsPattern": ".",  "endsPattern": "."  }  }  }  ]  } |

选择QEMU Debug x86 bios boot



选择QEMU Debug x86 bios kernel



# 适配新平台

## 增加启动代码

village-boot/xxxx/boot.s

## 增加框架平台

village-kernel/vklibs/libhw/xxx

## 增加初始化代码

village-kernel/vklibs/libhw/xxx/cpu/crt0/crt0\_kernel.cpp

## 增加链接信息

village-kernel/vklibs/libhw/xxx/cpu/lds

## 增加基本驱动

village-kernel/drivers/xxx/\*

## 增加时钟管理

village-kernel/arch/xxx/src/System.cpp

## 增加中断管理

village-kernel/arch/xxx/src/ArchInterrupt.cpp

village-kernel/arch/xxx/src/Exception.cpp

## 增加调度管理

village-kernel/arch/xxx/src/Scheduler.cpp

## 修改或增加Kconfig

village-kernel/Kconfig

village-kernel/vklibs/libhw/xxx/Kconfig

## 修改或增加Makefile

village-kernel/Makefile

village-kernel/vklibs/libhw/xxx/Makefile

# 编写驱动模块

驱动类型分为四种类型，分别为块设备block，网络设备network，字符设备character以及杂项设备misc。

|  |
| --- |
| //###########################################################################  // HelloDriver.cpp  // Definitions of the functions that manage hello driver  //  // $Copyright: Copyright (C) village  //###########################################################################  #include "Driver.h"  #include "Kernel.h"  /// @brief HelloDriver  class HelloDriver : public Driver  {  public:  /// @brief HelloDriver Open  void Open()  {  }  /// @brief HelloDriverClose  void Close()  {  }  };  //Register driver  REGISTER\_DRIVER(new HelloDriver(), DriverID::\_miscdev, helloDriver); |

# **编写功能模块**

功能模块分为三种类型，功能型feature，服务型serivce以及程序型program。

|  |
| --- |
| //###########################################################################  // HelloModule.cpp  // Definitions of the functions that manage hello module  //  // $Copyright: Copyright (C) village  //###########################################################################  #include "Module.h"  #include "Kernel.h"  /// @brief HelloModule  class HelloModule: public Module  {  public:  /// @brief HelloModule Setup  void Setup()  {  }  /// @brief HelloModule Exit  void Exit()  {  }  };  //Register module  REGISTER\_MODULE(new HelloModule(), ModuleID::\_feature, helloModule); |

# 编写应用程序

|  |
| --- |
| //###########################################################################  // HelloApp .cpp  // Definitions of the functions that manage hello app  //  // $Copyright: Copyright (C) village  //###########################################################################  /// @brief HelloApp  class HelloApp  {  public:  /// @brief HelloApp Initialize  void Initialize()  {  }  /// @brief HelloApp execute  void Execute()  {  }  };  /// @brief main  int main(void)  {  HelloApp app;  app.Initialize();  app.Execute();  return 0;  } |