Village Kernel 开发指南

从零开始写内核

溪江朔

VILALGE

目录

第一章：序言 4

一、 Village 内核特点：.. 4

二、 Village 内核目标：.. 4

三、 进展说明 4

四、 说明 5

第二章：搭建开发环境 6

一、 系统要求 6

二、 搭建开发环境 6

三、 克隆 village-kernel 项目 6

四、 使用 vscode 打开 village-kernel 项目 7

五、 创建 rootfs 文件系统镜像 7

六、 运行与调试代码 9

第三章：简易框架 10

一、 boot 引导层 10

二、 app 应用层 16

三、 debug 调试 23

第四章：内核结构 27

一、 village-boot：加载器代码 27

二、 village-kernel：微内核代码 27

三、 village-machine：硬件层代码 30

四、 village-demo：用户层代码 31

五、 village-scripts：工具脚本 31

六、 village-docs：相关文档 31

七、 build：编译输出文件夹 31

第五章：执行流程 32

一、 启动加载内核并跳转执行 32

二、 设置中断向量表 37

三、 初始化段数据和构造函数 50

四、 跳转到 main 函数 52

五、 初始化内核模块 53

六、 初始化芯片时钟 54

七、 初始化中断向量 55

八、 其他初始化略过 57

九、 初始化文件系统并挂载根目录 57

十、 加载共享库及动态模块 59

十一、 开始调度任务 60

十二、 加载第一个应用程序 62

十三、 应用层展开服务以及功能。 64

第六章：适配新平台 67

一、　增加启动代码......67
二、　增加框架平台......67
三、　增加初始化代码......67
四、　增加链接信息......67
五、　增加基本驱动......67
六、　增加时钟管理......67
七、　增加中断管理......67
八、　增加调度管理......68
九、　修改或增加 Kconfig......68
十、　修改或增加 Makefile......68
第七章：普通设备驱动......68
第八章：平台设备驱动......69
第九章：编写功能模块......72
第十章：编写应用程序......73
第十一章：内核 API 调用......74

第一章：序言

一、 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

二、 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

三、 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：

- 适配更多平台
- 其他文件系统
- 网络功能

四、 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。

第二章：搭建开发环境

一、 系统要求

mac os / linux / windows（使用 wsl 子系统）

二、 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
```

三、 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
```

四、 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
```

五、 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
```

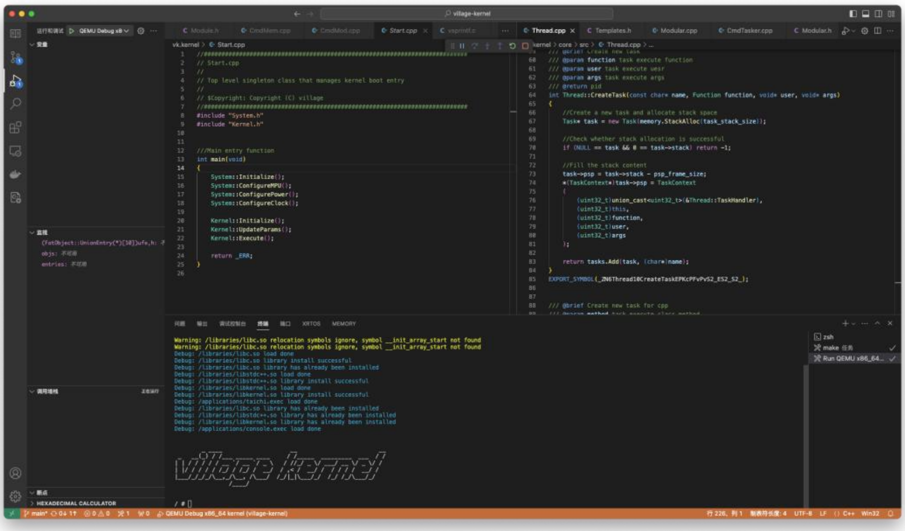
拷贝相关文件到文件系统

```
sudo make rootfs
```

六、 运行与调试代码

切换到 vscode debug 界面

选择 QEMU Debug x86_64 kernel



```
//###########################################################################
// Start.cpp
//
// Top level singleton class that manages kernel boot entry
//
// $Copyright: Copyright (C) village
//###########################################################################
#include "System.h"
#include "Kernel.h"


///Main entry function
int main(void)
{
    System::Initialize();
    System::ConfigureMPU();
    System::ConfigurePower();
    System::ConfigureClock();

    Kernel::Initialize();
    Kernel::UpdateParams();
    Kernel::Execute();

    return _ERR;
}
```

```
/// @brief Create new task
/// @param function task execute function
/// @param user task execute uesr
/// @param args task execute args
/// @return pid
int Thread::CreateTask(const char* name, Function function, void* user, void* args)
{
    //Create a new task and allocate stack space
    Task* task = new Task(memory.StackAlloc(task_stack_size));

    //Check whether stack allocation is successful
    if (NULL == task && 0 == task->stack) return -1;

    //Fill the stack content
    task->psp = task->stack - psp_frame_size;
    *(TaskContext*)task->psp = TaskContext
    {
        (uint32_t)union_cast<uint32_t>(&Thread::TaskHandler),
        (uint32_t)this,
        (uint32_t)function,
        (uint32_t)user,
        (uint32_t)args
    };

    return tasks.Add(task, (char*)name);
}
EXPORT_SYMBOL(_ZN6Thread10CreateTaskEPKcPFvPvS2_ES2_S2_);


/// @brief Create new task for cpp
```

```
Warning: /libraries/libc.so relocation symbols ignore, symbol __init_array_start not found
Warning: /libraries/libc.so relocation symbols ignore, symbol __init_array_start not found
Debug: /libraries/libc.so load done
Debug: /libraries/libc.so library install successful
Debug: /libraries/libc.so library has already been installed
Debug: /libraries/libstdc++.so load done
Debug: /libraries/libstdc++.so library install successful
Debug: /libraries/libkernel.so load done
Debug: /libraries/libkernel.so library install successful
Debug: /applications/taichi.exec load done
Debug: /libraries/libc.so library has already been installed
Debug: /libraries/libstdc++.so library has already been installed
Debug: /libraries/libkernel.so library has already been installed
Debug: /applications/console.exec load done

village kernel

/ # 
```

第三章：简易框架

一、 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
```

二、 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
```

三、 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 ..
```

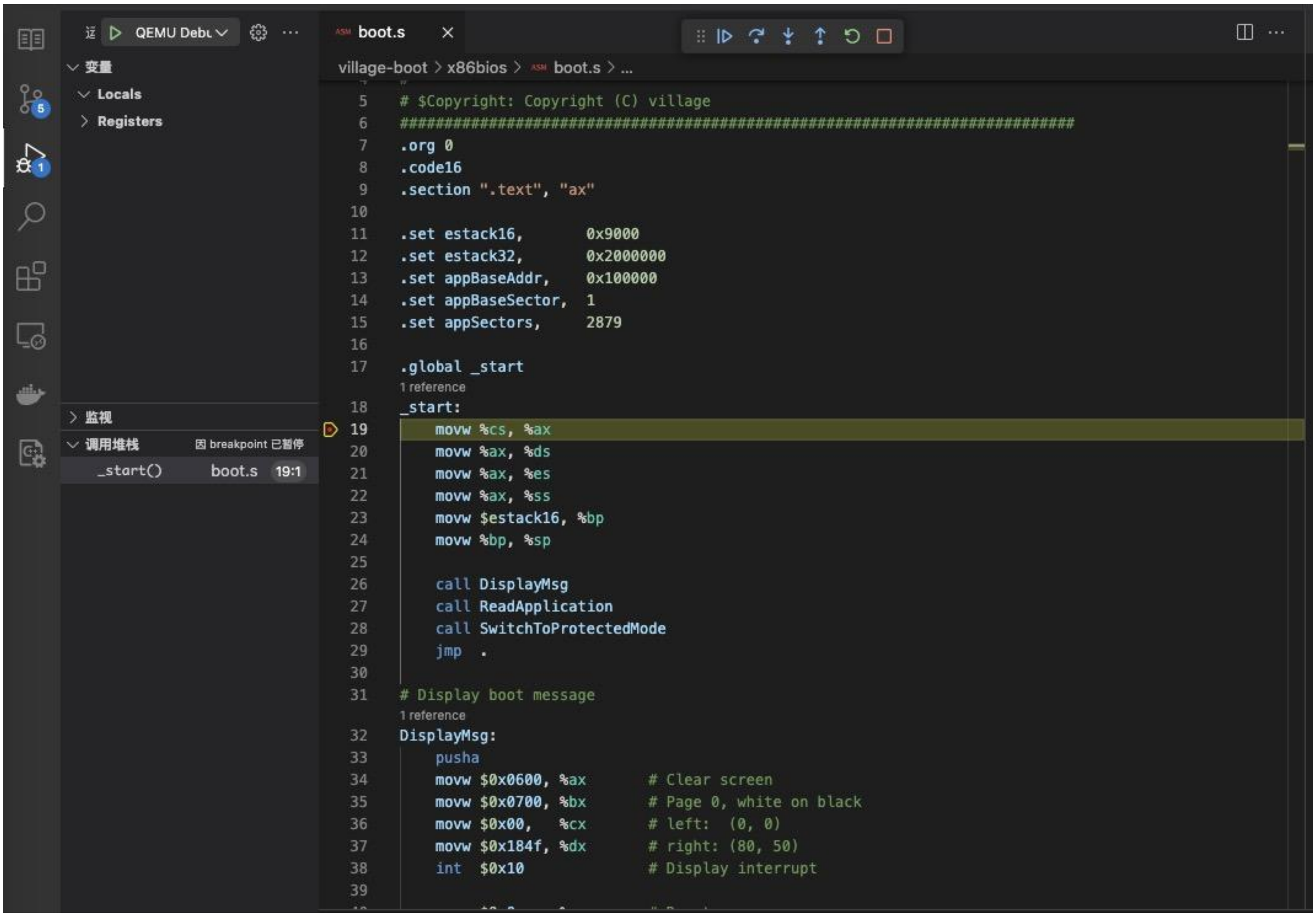
01_boot/.vscode/launch.json

```
{
	// 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

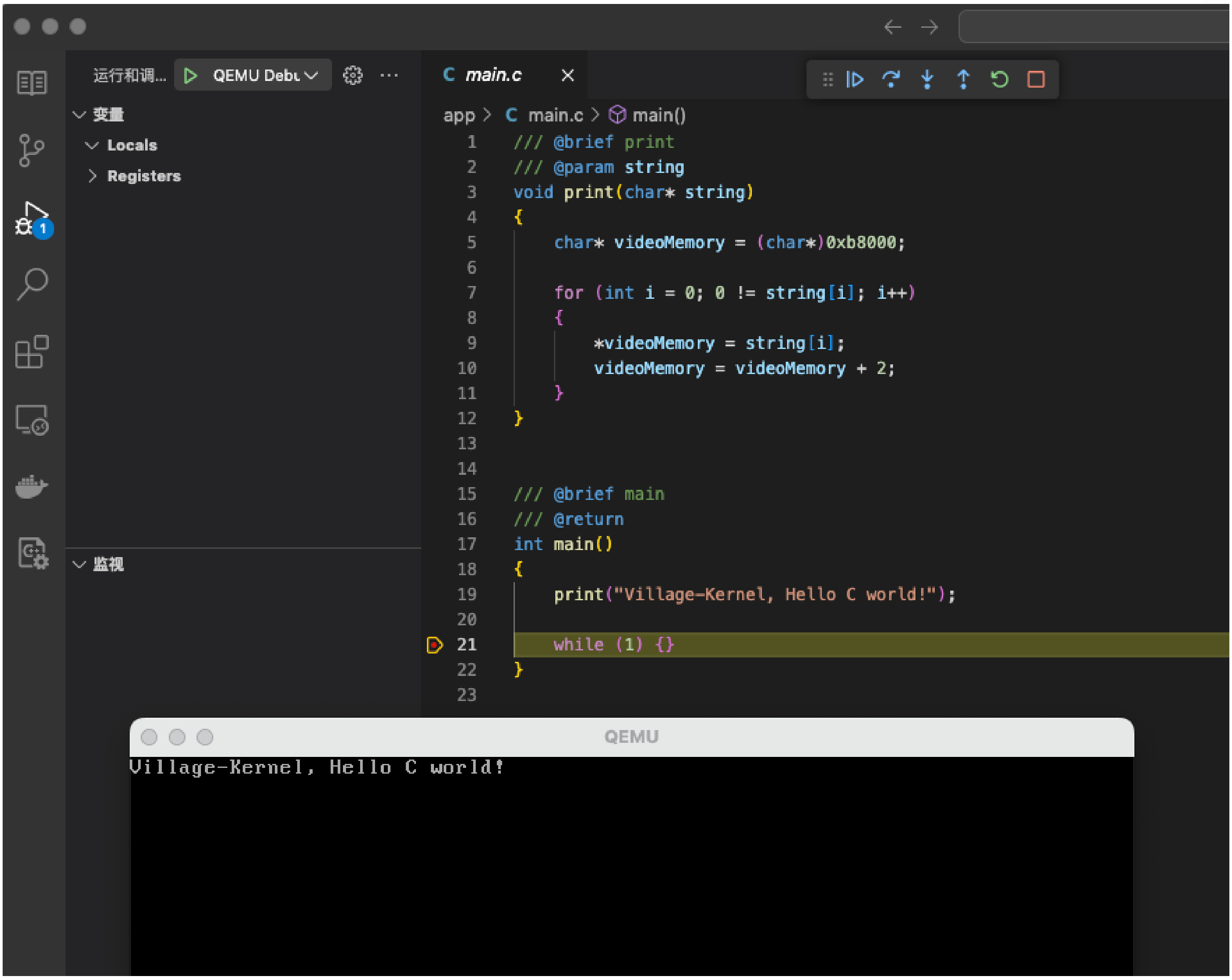
```
{
    "version": "2.0.0",
    "tasks": [
        {
            "type":"shell",
            "label": "Build",
            "command": "make",
            "args": [
                "all"
            ],
            "detail": "Build project"
        },
        {
            "label": "Run QEMU x86 bios",
            "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



```
# $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

```

选择 QEMU Debug x86 bios kernel



```
/// @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) {}
}

```

```
Village-Kernel, Hello C world!
```

第四章：内核结构

一、 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。

二、 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 village-machine 目录下。

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，动态模块工具。

misc：

该目录下的代码会归类的 libmisc.so 共享库。

lock：

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。

stream：

存放模型相关代码。

stream：

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。

template：

这里存放一些模板类代码。

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。

terminal：

串口终端相关代码。

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。

三、 village-machine: 硬件层代码

适配新平台时，主要工作都在此目录，用来存放与机器相关的代码。

board:

存放具体机器的配置文件

chipdrv:

存放芯片模块驱动代码

platdrv:

存放外围设备驱动代码

vendor:

该目录下的代码会归类到 libhw.so 共享库。
定义编译时的相关参数以及链接时的相关参数。
存放厂家提供的底层代码，包括启动代码，链接文件等。

四、 village-demo: 用户层代码

libaries：

存放一些代码库。

modules：

存放内核模块代码。

programs：

存放应用程序代码。

services：

存放服务程序代码。

五、 village-scripts: 工具脚本

存放工具脚本，menuconfig 和 configs 存放在该目录

六、 village-docs：相关文档

存放 village 的相关文档。

七、 build：编译输出文件夹

编译生成的临时文件存放目录。

第五章：执行流程

一、 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/legacy/ia32bios/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
```

二、 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-machine/vendor/ia32legacy/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,
};
```

三、 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-machine/vendor/ia32legacy/crt0/crt0_kernel.c

```
//################################################################################
// crt0_kernel.c
// Low level file that manages kernel entry
//
// $Copyright: Copyright (C) village
//################################################################################


/// @brief KernelSymbol
void KernelSymbol();


/// @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();

    KernelSymbol();

    __preinit_arrary();

    __init_array();

    main(argc, argv);

    __fini_array();

    for(;;) ;
}
```

四、 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/vk_village.cpp

```
/// @brief Definition kernel
Kernel* kernel = NULL;


/// @brief Export kernel symbol, call by crt0 _start function
extern "C" void KernelSymbol()
{
    if (NULL == kernel)
    {
        kernel = &Village::Instance();
        kernel->symbol.Export((uint32_t)&kernel, "kernel");
    }
}


/// @brief Main entry function
/// @param argc
/// @param argv
/// @return
int main(int argc, char* argv[])
{
    kernel->Setup();
    kernel->Start();
    kernel->Exit();
    return 0;
}
```

五、 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/vk_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();
}
```

六、 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/ia32/legacy/src/vk_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++; }
```

七、 初始化中断向量

village-kernel/arch/ia32/legacy/src/vk_exception.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);
}
```

八、 其他初始化略过

九、 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");
}
```

十、 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;
}
```

十一、　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/ia32/legacy/src/vk_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");
}
```

十二、 加载第一个应用程序

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

village-kernel/kernel/src/vk_process.cpp

```
void ConcreteProcess::Taichi()
{
	const char* taichi = "/services/taichi.exec";

	if (0 > Run(Process::_Background, taichi))
	{
		kernel->debug.Error("%s execute failed", taichi);
	}
}
```

village-kernel/binutils/src/vk_base_executor.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/binutils/src/vk_elf_executor.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/binutils/src/vk_elf_loader.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
```

十三、　应用层展开服务以及功能。

内核调用并执行 taichi.exec 之后，则来到了应用层，taichi 服务会加载其他服务和程序。

village-demo/services/taichi/src/taichi_service.cpp

```
//#################################################################################
// Taichi.cpp
// Definitions of the functions that manage taichi service
//
// $Copyright: Copyright (C) village
```

```
//#################################################################################
#include "taichi_service.h"
#include "vk_rc_parser.h"
#include "vk_kernel.h"


/// @brief Constructor
TaichiService::TaichiService()
{
}


/// @brief Destructor
TaichiService::~TaichiService()
{
}


/// @brief Loader load
/// @param filename rc file path
void TaichiService::Load(const char* filename)
{
	RcParser* parser = new RcParser(filename);

	VkList<char*>& runcmds = parser->GetRunCmds();

	for (runcmds.Begin(); !runcmds.IsEnd(); runcmds.Next())
	{
		kernel->process.Run(Process::_Background, runcmds.Item());
	}

	parser->Release();
	delete parser;
}


/// @brief Loader unload
/// @param filename rc file path
void TaichiService::Unload(const char* filename)
{
	RcParser* parser = new RcParser(filename);

	VkList<char*>& runcmds = parser->GetRunCmds();
```

```
        for (runcmds.End(); !runcmds.IsBegin(); runcmds.Prev())
        {
                kernel->process.Kill(runcmds.Item());
        }

        parser->Release();
        delete parser;
}


/// @brief Setup
void TaichiService::Setup()
{
        //Load services
        Load("/services/_load_.rc");

        //Load programs
        Load("/programs/_load_.rc");
}


/// @brief Execute
void TaichiService::Execute()
{
        kernel->thread.Blocked();
}


/// @brief Exit
void TaichiService::Exit()
{
        //Unload programs
        Unload("programs/_load_.rc");

        //Unload services
        Unload("services/_load_.rc");
}


/// @brief main
int main(void)
{
        TaichiService taichi;
        taichi.Setup();
```

```
	taichi.Execute();
	taichi.Exit();
	return 0;
}
```

第六章：适配新平台

一、　增加启动代码

village-boot/xxxx/boot.s

二、　增加框架平台

village-machine/vendor/xxx

三、　增加初始化代码

village-machine/vendor/xxx/crt0/crt0_kernel.c

四、　增加链接信息

village-machine/vendor/xxx/lds/kernel.lds

五、　增加基本驱动

village-machine/platdrv/xxx/*

六、　增加时钟管理

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

七、　增加中断管理

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

八、　增加调度管理

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

九、　修改或增加 Kconfig

village-machine/Kconfig
village-machine/vendor/xxx/Kconfig

十、　修改或增加 Makefile

village-machine/Makefile
village-machine/vendor/xxx/Makefile

第七章：普通设备驱动

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 MiscDriver
{
public:
	/// @brief HelloDriver Open
	void Open()
	{

	}

	/// @brief HelloDriverClose
	void Close()
	{
	}
};



//Register driver
REGISTER_MISC_DEVICE(new HelloDriver(), helloDriver);
```

第八章：平台设备驱动

平台设备驱动，用来分离设备和驱动，让驱动代码具有可复用性。可以简单理解，就是把具体设备的配置放到 device 层，分离设备具体配置，而具体配置可通过 Config 结构体修改。一个 device 层只能对应一个驱动，而一个 driver 则能对应多个 device。驱动与设备之间的匹配目前只有名称（在注册驱动和设备时，使用的宏，第二个参数）匹配方式。具体看例程代码。

平台设备驱动层代码。

```
#include "vk_kernel.h"



/// @brief HelloPlatform
class HelloPlatform : public MiscDriver
{
public:
	//Config
	struct Config
	{
		uint8_t data;
	};
private:
	/// @brief Members
	Config config;
public:
	/// @brief Constructor
	HelloPlatform()
	{

	}

	/// @brief Desturctor
	~HelloPlatform()
	{

    }

    /// @brief Plat Methods
    void SetData(void* data)
    {
        config = *((Config*)data);
    }

    /// @brief Fopts Methods
    bool Open()
    {

    }

    /// @brief Write
    /// @param data
    /// @param count
    /// @param blk
    /// @return
    int Write(uint8_t* data, uint32_t count, uint32_t blk)
    {
        return 0;
    }

    /// @brief Read
    /// @param data
    /// @param count
    /// @param blk
    /// @return
    int Read(uint8_t* data, uint32_t count, uint32_t blk)
    {
        return 0;
    }

    /// @brief Close
    void Close()
    {

    }
};


/// @brief HelloPlatDrv
class HelloPlatDrv : public PlatDriver
```

```
{
public:
    /// @brief Probe
    bool Probe(PlatDevice* device)
    {
        device->Attach(new HelloPlatform());
        kernel->device.RegisterMiscDevice((MiscDriver*)device->GetDriver());
        return true;
    }

    /// @brief Remove
    /// @param device
    /// @return
    bool Remove(PlatDevice* device)
    {
        kernel->device.UnregisterMiscDevice((MiscDriver*)device->GetDriver());
        delete (HelloPlatform*)device->GetDriver();
        device->Detach();
        return true;
    }
};


///Register driver
REGISTER_PLAT_DRIVER(new HelloPlatDrv(), helloplatform, helloPlatDrv);
```

平台设备设备层

```
#include "vk_kernel.h"

/// @brief HelloPlatDev1
class HelloPlatDev1 : public PlatDevice
{
private:
    /// @brief Members
    HelloPlatform::Config config;
public:
    /// @brief Methods
    void Config()
    {
        config = {
            .data = 1,
        };
        driverData = (void*)&config;
        driverName = (char*)"hello plat 1";
    }
};

///Register device
REGISTER_PLAT_DEVICE(new HelloPlatDev1(), helloplatform, helloPlatDev1);



/// @brief HelloPlatDev2
class HelloPlatDev2 : public PlatDevice
{
private:
    /// @brief Members
    HelloPlatform::Config config;
public:
    /// @brief Methods
    void Config()
    {
        config = {
            .data = 2,
        };
        driverData = (void*)&config;
        driverName = (char*)"hello plat 2";
    }
};

///Register device
REGISTER_PLAT_DEVICE(new HelloPlatDev2(), helloplatform, helloPlatDev2);
```

第九章：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
	bool Setup()
	{

	}

	/// @brief HelloModule Exit
	void Exit()
	{

	}
};



//Register module
REGISTER_MODULE(new HelloModule(), ModuleID::_feature, helloModule);
```

第十章：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;
}
```

第十一章：内核 API 调用

内核 api 调用头文件: village-kernel/kernel/if/vk_kernel.h

```
//############################################################################
// vk_kernel.h
// Declarations of the village interface
//
// $Copyright: Copyright (C) village
//############################################################################
#ifndef __VK_KERNEL_H__
#define __VK_KERNEL_H__

#include "stdint.h"
#include "stddef.h"
#include "stdlib.h"
#include "vk_list.h"
#include "vk_driver.h"
#include "vk_module.h"

/// @brief System
class System
{
```

```
public:
    /// @brief Ticks Methods
    virtual void SysTickCounter() = 0;
    virtual uint32_t GetSysClkCounts() = 0;
    virtual void DelayMs(uint32_t millis) = 0;

    /// @brief IRQ Methods
    virtual void EnableIRQ() = 0;
    virtual void DisableIRQ() = 0;

    /// @brief Power Methods
    virtual void Sleep() = 0;
    virtual void Standby() = 0;
    virtual void Shutdown() = 0;
    virtual void Reboot() = 0;
};

/// @brief Memory
class Memory
{
public:
    /// @brief Alloc Methods
    virtual uint32_t HeapAlloc(uint32_t size) = 0;
    virtual uint32_t StackAlloc(uint32_t size) = 0;
    virtual void Free(uint32_t memory, uint32_t size = 0) = 0;

    /// @brief Info Methods
    virtual uint32_t GetSize() = 0;
    virtual uint32_t GetUsed() = 0;
    virtual uint32_t GetCurrAddr() = 0;
};

/// @brief Debug
class Debug
{
public:
    /// @brief Debug level
    enum Level
    {
        _Lv0 = 0,
        _Lv1,
        _Lv2,
        _Lv3,
        _Lv4,
                _Lv5
        };
public:
        /// @brief Methods
        virtual void Log(const char* format, ...) = 0;
        virtual void Info(const char* format, ...) = 0;
        virtual void Error(const char* format, ...) = 0;
        virtual void Warn(const char* format, ...) = 0;
        virtual void Output(int level, const char* format, ...) = 0;
        virtual void SetDebugLevel(int level) = 0;
};

/// @brief Interrupt
class Interrupt
{
public:
        /// @brief Set Methods
        virtual int SetISR(int irq, Function func, void* user = NULL, void* args = NULL) = 0;
        virtual int SetISR(int irq, Method method, Class* user, void* args = NULL) = 0;

        /// @brief Append Methods
        virtual int AppendISR(int irq, Function func, void* user = NULL, void* args = NULL) = 0;
        virtual int AppendISR(int irq, Method method, Class* user, void* args = NULL) = 0;

        /// @brief Remove Methods
        virtual bool RemoveISR(int irq, Function func, void* user = NULL, void* args = NULL) = 0;
        virtual bool RemoveISR(int irq, Method method, Class* user, void* args = NULL) = 0;

        /// @brief Clear Methods
        virtual void ClearISR(int irq) = 0;

        /// @brief Replace Methods
        virtual void Replace(int irq, uint32_t handler) = 0;

        /// @brief Feature Methods
        virtual void Handler(int irq) = 0;
};

/// @brief Scheduler
class Scheduler
{
public:
        /// @brief Methods
        virtual void Start() = 0;
```

```
    virtual void Sched() = 0;
};

/// @brief Thread
class Thread
{
public:
    /// @brief Enumerations
    enum TaskState
    {
        _Pending = 0,
        _Running,
        _Suspend,
        _Blocked,
        _Exited,
    };

    /// @brief Structures
    struct Task
    {
        char*                  name;
        uint32_t              tid;
        uint32_t              psp;
        uint32_t              ticks;
        uint32_t              stack;
        TaskState            state;

        Task(uint32_t stack = 0, char* name = NULL)
            :name(name),
            tid(-1),
            psp(0),
            ticks(0),
            stack(stack),
            state(TaskState::_Pending)
        {}
    };
public:
    /// @brief Create Methods
    virtual int CreateTask(const char* name, Function function, void* user = NULL, void* args = NULL) = 0;
    virtual int CreateTask(const char* name, Method method, Class *user, void* args = NULL) = 0;

    /// @brief Task Methods
    virtual int GetTaskId() = 0;
    virtual bool StartTask(int tid) = 0;
```

```
        virtual bool StopTask(int tid) = 0;
        virtual bool WaitForTask(int tid) = 0;
        virtual bool ExitBlocked(int tid) = 0;
        virtual bool DeleteTask(int tid) = 0;
        virtual bool IsTaskAlive(int tid) = 0;
        virtual VkList<Task*> GetTasks() = 0;

        /// @brief State Methods
        virtual void ChangeState(TaskState state) = 0;
        virtual void Sleep(uint32_t ticks) = 0;
        virtual void Blocked() = 0;
        virtual void TaskExit() = 0;

        /// @brief Scheduler Methods
        virtual void SaveTaskPSP(uint32_t psp) = 0;
        virtual uint32_t GetTaskPSP() = 0;
        virtual void SelectNextTask() = 0;
        virtual void IdleTask() = 0;
};

/// @brief Symbol
class Symbol
{
public:
        /// @brief Methods
        virtual void Export(uint32_t symAddr, const char* name) = 0;
        virtual void Unexport(const char* name) = 0;
        virtual uint32_t Search(const char* name) = 0;
};

/// @brief Device
class Device
{
public:
        /// @brief Block device methods
        virtual void RegisterBlockDevice(BlockDriver* driver) = 0;
        virtual void UnregisterBlockDevice(BlockDriver* driver) = 0;

        /// @brief Char device methods
        virtual void RegisterCharDevice(CharDriver* driver) = 0;
        virtual void UnregisterCharDevice(CharDriver* driver) = 0;

        /// @brief Framebuffer device methods
        virtual void RegisterFBDevice(FBDriver* driver) = 0;
```

```
	virtual void UnregisterFBDevice(FBDriver* driver) = 0;

	/// @brief Input device methods
	virtual void RegisterInputDevice(InputDriver* driver) = 0;
	virtual void UnregisterInputDevice(InputDriver* driver) = 0;

	/// @brief Netwrok device methods
	virtual void RegisterNetworkDevice(NetworkDriver* driver) = 0;
	virtual void UnregisterNetworkDevice(NetworkDriver* driver) = 0;

	/// @brief Misc device methods
	virtual void RegisterMiscDevice(MiscDriver* driver) = 0;
	virtual void UnregisterMiscDevice(MiscDriver* driver) = 0;

	/// @brief Platform device methods
	virtual void RegisterPlatDevice(PlatDevice* device) = 0;
	virtual void UnregisterPlatDevice(PlatDevice* device) = 0;

	/// @brief Platform driver methods
	virtual void RegisterPlatDriver(PlatDriver* driver) = 0;
	virtual void UnregisterPlatDriver(PlatDriver* driver) = 0;

	/// @brief Data methods
	virtual Fopts* GetDeviceFopts(const char* name) = 0;
	virtual VkList<Base*> GetDevices(DriverID id) = 0;
};

/// @brief Feature
class Feature
{
public:
	/// @brief Register Methods
	virtual void RegisterModule(Module* module) = 0;
	virtual void UnregisterModule(Module* module) = 0;

	/// @brief Data Methods
	virtual Module* GetModule(const char* name) = 0;
};

/// @brief FileSys
class FileSys;

/// @brief FileVol
class FileVol;
```

```
/// @brief FileSystem
class FileSystem
{
public:
    /// @brief Hard Drive Methods
    virtual bool MountHardDrive(const char* disk) = 0;
    virtual bool UnmountHardDrive(const char* disk) = 0;

    /// @brief Register Methods
    virtual void RegisterFS(FileSys* fs, const char* name) = 0;
    virtual void UnregisterFS(FileSys* fs, const char* name) = 0;

    /// @brief Volume Methods
    virtual FileVol* GetVolume(const char* name) = 0;
};

/// @brief WorkQueue
class WorkQueue
{
public:
    /// @brief Enumerations
    enum State
    {
        _Suspend = 0,
        _Waked,
        _Running,
        _Finish,
    };

    /// @brief Structures
    struct Work
    {
        Function func;
        void*    user;
        void*    args;
        uint32_t ticks;
        State    state;

        Work(Function func, void* user, void* args, uint32_t ticks)
            :func(func),
            user(user),
            args(args),
            ticks(ticks),
            state(_Suspend)
        {}
    };
public:
    /// @brief Create Methods
    virtual Work* Create(Function func, void* user = NULL, void* args = NULL, uint32_t ticks = 0) = 0;
    virtual Work* Create(Method method, Class* user, void* args = NULL, uint32_t ticks = 0) = 0;

    /// @brief Feature Methods
    virtual bool Delete(Work* work) = 0;
    virtual bool Sched(Work* work) = 0;
};

/// @brief Event
class Event
{
public:
    /// @brief Types
    enum EventType
    {
        _InputKey = 0,
        _InputAxis,
        _OutputText,
        _OutputAxis,
        _AllType,
    };

    /// @brief Output format
    enum OutFormat
    {
        _Noraml,
        _Terminal,
    };

    /// @brief Input key
    struct InputKey
    {
        int code;
        int status;

        InputKey()
            :code(0),
            status(0)
        {}
    };

    /// @brief Input axis
    struct InputAxis
    {
        int axisX;
        int axisY;
        int axisZ;

        InputAxis()
            :axisX(0),
            axisY(0),
            axisZ(0)
        {}
    };

    /// @brief Output text
    struct OutputText
    {
        char* data;
        int   size;

        OutputText()
            :data(NULL),
            size(0)
        {}
    };

    /// @brief Ouput Axis
    struct OutputAxis
    {
        int axisX;
        int axisY;
        int axisZ;

        OutputAxis()
            :axisX(0),
            axisY(0),
            axisZ(0)
        {}
    };
public:
    /// @brief Device Methods
    virtual void InitInputDevice(const char* input) = 0;
```

```
        virtual void ExitInputDevice(const char* input) = 0;

        /// @brief Attach Methods
        virtual void Attach(EventType type, Method method, Class* user) = 0;
        virtual void Attach(EventType type, Function func, void* user = NULL) = 0;
        virtual void Detach(EventType type, Method method, Class* user) = 0;
        virtual void Detach(EventType type, Function func, void* user = NULL) = 0;

        /// @brief Input Methods
        virtual void ReportKey(int code, int status) = 0;
        virtual void ReportAxis(int axisX, int axisY, int axisZ) = 0;

        /// @brief Output Methods
        virtual void PushChar(char chr) = 0;
        virtual void PushString(char* data, int size) = 0;
        virtual void PushAxis(int axisX, int axisY, int axisZ) = 0;
        virtual void SetOutFormat(OutFormat format) = 0;
        virtual OutFormat GetOutFormat() = 0;
};

/// @brief ElfLoader
class ElfLoader;

/// @brief Loader
class Loader
{
public:
        /// @brief Enumerations
        enum LoadType
        {
                _Lib = 0,
                _Mod,
        };
public:
        /// @brief Load Methods
        virtual void Load(int type, const char* filename) = 0;
        virtual void Unload(int type, const char* filename) = 0;

        /// @brief Install Methods
        virtual bool Install(int type, const char* filename) = 0;
        virtual bool Uninstall(int type, const char* filename) = 0;

        /// @brief Data Methods
        virtual VkList<ElfLoader*>* GetLibraries() = 0;
        virtual VkList<ElfLoader*>* GetModules() = 0;
};

/// @brief Executor
class Executor;

/// @brief BaseExecutor
class BaseExecutor;

/// @brief Process
class Process
{
public:
        /// @brief Enumerations
        enum Behavior
        {
                _Foreground = 0,
                _Background = 1,
        };

        /// @brief Structures
        struct Data
        {
                char*               name;
                int                 pid;
                int                 tid;
                BaseExecutor* exec;

                Data(char* name = NULL)
                        :name(name),
                        pid(0),
                        tid(0),
                        exec(NULL)
                {}
        };
public:
        /// @brief Register Methods
        virtual void RegisterExecutor(Executor* executor) = 0;
        virtual void UnregisterExecutor(Executor* executor) = 0;

        /// @brief Run Methods
        virtual int Run(Behavior behavior, const char* args) = 0;
        virtual int Run(Behavior behavior, const char* path, int argc, char* argv[]) = 0;
```

```
    /// @brief Kill Methods
    virtual bool Kill(const char* path) = 0;
    virtual bool Kill(int pid) = 0;

    /// @brief Check Methods
    virtual bool IsExist(const char* path) = 0;
    virtual bool IsExist(int pid) = 0;

    /// @brief Data Methods
    virtual VkList<Data*> GetData() = 0;
};

/// @brief Timer
class Timer
{
public:
    /// @brief Enumerations
    enum State
    {
        _Ready = 0,
        _Done,
    };

    /// @brief Structures
    struct Job
    {
        Function func;
        void*    user;
        void*    args;
        uint32_t ticks;
        State    state;

        Job(uint32_t ticks, Function func, void* user, void* args)
            :func(func),
            user(user),
            args(args),
            ticks(ticks),
            state(_Ready)
        {}
    };
public:
    /// @brief Create Methods
    virtual Job* Create(uint32_t ticks, Function func, void* user = NULL, void* args = NULL) = 0;
    virtual Job* Create(uint32_t ticks, Method method, Class* user, void* args = NULL) = 0;
```

```
        /// @brief Feature Methods
        virtual void Modify(Job* job, uint32_t ticks) = 0;
        virtual bool Delete(Job* job) = 0;
};

/// @brief Cmd
class Cmd;

/// @brief Console
class Console;

/// @brief Terminal
class Terminal
{
public:
        /// @brief Structures
        struct Sandbox
        {
                int             cid;
                int             tid;
                char*           driver;
                Console*        console;

                Sandbox(char* driver = NULL)
                        :cid(0),
                        tid(0),
                        driver(driver),
                        console(NULL)
                {}
        };
public:
        /// @brief Cmd Methods
        virtual void RegisterCmd(Cmd* cmd, char* name) = 0;
        virtual void UnregisterCmd(Cmd* cmd, char* name) = 0;
        virtual VkList<Cmd*> GetCmds() = 0;

        /// @brief Console Methods
        virtual int CreateConsole(const char* driver) = 0;
        virtual bool DestroyConsole(const char* driver) = 0;
        virtual VkList<Sandbox*> GetSandboxes() = 0;
};

/// @brief Signal
```

```
class Signal
{
public:
	/// @brief Feature Methods
	virtual void Raising(int signal) = 0;
};

/// @brief Stack
class Stack;

/// @brief Protocol
class Protocol
{
public:
	/// @brief Stack Methods
	virtual void RegisterStack(Stack* stack) = 0;
	virtual void UnregisterStack(Stack* stack) = 0;
};

/// @brief Kernel
class Kernel
{
public:
	/// @brief Members
	System&        system;
	Memory&        memory;
	Debug&         debug;
	Interrupt&     interrupt;
	Scheduler&     scheduler;
	Thread&        thread;
	WorkQueue&     workQueue;
	Event&         event;
	Symbol&        symbol;
	Device&        device;
	Feature&       feature;
	FileSystem&    filesys;
	Loader&        loader;
	Process&       process;
	Timer&         timer;
	Terminal&      terminal;
	Signal&        signal;
	Protocol&      protocol;
public:
	/// @brief constructor
```

```
Kernel(
        System&         system,
        Memory&         memory,
        Debug&          debug,
        Interrupt&      interrupt,
        Scheduler&      scheduler,
        Thread&         thread,
        WorkQueue&      workQueue,
        Event&          event,
        Symbol&         symbol,
        Device&         device,
        Feature&        feature,
        FileSystem&     filesys,
        Loader&         loader,
        Process&        process,
        Timer&          timer,
        Terminal&       terminal,
        Signal&         signal,
        Protocol&       protocol
)
        :system(system),
        memory(memory),
        debug(debug),
        interrupt(interrupt),
        scheduler(scheduler),
        thread(thread),
        workQueue(workQueue),
        event(event),
        symbol(symbol),
        device(device),
        feature(feature),
        filesys(filesys),
        loader(loader),
        process(process),
        timer(timer),
        terminal(terminal),
        signal(signal),
        protocol(protocol)
{}

/// @brief Destructor
virtual ~Kernel() {}

/// @brief Kernel Methods
```

```
        virtual void Setup() = 0;
        virtual void Start() = 0;
        virtual void Exit() = 0;

        /// @brief Power Methods
        virtual void Sleep() = 0;
        virtual void Standby() = 0;
        virtual void Shutdown() = 0;
        virtual void Reboot() = 0;

        /// @brief Kernel build info
        virtual const char* GetBuildDate() = 0;
        virtual const char* GetBuildTime() = 0;
        virtual const char* GetBuildVersion() = 0;
        virtual const char* GetBuildGitCommit() = 0;
};

/// @brief Declarations kernel pointer
extern Kernel* kernel;

#endif //!__VK_KERNEL_H__
```