CMake

mkdir build

cmake <CMakeLists.txt对应的文件路径>

make

·指定使用的最低cmake版本

**cmake\_minimum\_required**(VERSION 3.12)

·定义工程名称

**project**(test)

·生成一个可执行程序

**add\_executable**(app aaa.c bbb.c ccc.c ddd.c)

·定义变量

**set**(SRC\_LIST add.c div.c mult.c sub.c main.c)

**add\_executable**(app ${SRC\_LIST})

·指定C++标准

**set**(CMAKE\_CXX\_STANDARD 11)

·指定可执行程序输出路径

**set**(HOME /home/robin/linux/sort)

**set**(EXECUTABLE\_OUTPUT\_PATH ${HOME}/bin)

·搜索文件

**aux\_source\_directory**(<dir> <variable>)

dir: 要搜索的目录

variable: 将dir目录下搜索到的源文件列表存储到该变量中

**file**(GLOB/GLOB\_RECURSE 变量名 要搜索的文件路径和文件类型)

GLOB: 将指定目录下搜索到的满足条件的所有文件名生成一个列表，存储到变量

GLOB\_RECURSE: 递归搜索指定目录

·指定头文件路径

**include\_directories**(头文件路径)

**PROJECT\_SOURCE\_DIR**

指向 最近一次调用project()命令的 CMakeLists.txt 文件 所在的目录。

在包含多个子项目的情况下，每个子项目（每次调用project()命令）都会重新设置 PROJECT\_SOURCE\_DIR，这使得每个子项目可以有自己独立的源代码目录路径。

**CMAKE\_SOURCE\_DIR**

指向最顶层的 CMakeLists.txt 文件所在的目录，即整个项目的根目录。

这个变量在整个项目构建过程中保持不变，无论当前处理的是哪个 CMakeLists.txt 文件。用于定义整个项目范围内的源代码路径。

**CMAKE\_CURRENT\_SOURCE\_DIR**

**---------**

**PROJECT\_BINARY\_DIR**

构建目录

**CMAKE\_BINARY\_DIR**

构建目录

**CMAKE\_CURRENT\_BINARY\_DIR**

·静态库

**add\_library**( libxxx.a STATIC <源文件….>)

·动态库

**add\_library**( libxxx.so SHARED <源文件….>)

·设置动态库/静态库生成路径

**set**(LIBRARY\_OUTPUT\_PATH <path>)

·链接静态库

指定库路径

**link\_directories**(${PROJECT\_SOURCE\_DIR}/lib)

指定静态库

**link\_libraries**(libxxx.a libyyy.a libzzz.a)

**link\_libraries**(xxx yyy zzz)

·链接动态库

**target\_link\_libraries**( <target> libxxx.so PRIVATE libyyy.so INTERFACE libzzz.so)

target: 可执行程序 或 库

动态库的链接具有传递性，默认PUBLIC。

如D🡪A🡪B，C

若A🡪 PRIVATE B，则D🡪A时，D不能使用B中的数据

若A🡪C，则D🡪A时，D是可以知道C中的数据的

若A🡪INTERFACE B, INTERFACE C，A中可以使用B，C中的函数，但不知道使用的函数是B中的还是C中的。

·添加子CMake

**add\_subdirectory**(source\_dir )

内核驱动基本模块 **struct kobject**

kobject kset

**struct cdev**

字符设备结构体

dev\_t 32位数据类型 高12bit为主设备号，低20bit为次设备号

MAJOR(dev\_t)

MINOR(dev\_t)

MKDEV(ma, mi)

1、申请设备号

静态分配设备号：

事先知道主、次设备号时使用 cat /proc/devices

int **register\_chrdev\_region**(dev\_t from, unsigned count, const char \*name)

from：可以通过MKDEV(ma, mi)来指定

动态申请设备号：

int **alloc\_chrdev\_region**(dev\_t \*dev, unsigned baseminor, unsigned count, const char \*name)

2、初始化cdev结构体

void **cdev\_init**(struct cdev \*cdev, const struct file\_operations \*ops)

3、将初始化后的cdev结构体注册到内核中

int **cdev\_add**(struct cdev \*cdev, dev\_t dev, unsigned count)

4、创建设备类和设备节点

struct class \***class\_create**(struct module \*owner, const char \*name) /sys/class/目录下

struct device \***device\_create**(struct class \*class, /dev/目录下

struct device \*parent,

dev\_t devt,

void \*drvdata,

const char \*fmt, ...)

5、删除注销

void **class\_destroy**(struct class \*class);

void **device\_destroy**(struct class \*class, dev\_t dev)

void **unregister\_chrdev\_region**(dev\_t from, unsigned count)

void **cdev\_del**(struct cdev \*cdev)

创建一个设备文件**mknod** 如：mknod /dev/demo **c** 251 0 主设备号251次设备号0

一切皆文件

内核模块

#include <linux/init.h>

#include <linux/module.h>

#include <linux/kernel.h>

#define **KERN\_EMERG** KERN\_SOH "0" /\* 系统崩溃 \*/

#define **KERN\_ALERT** KERN\_SOH "1" /\* 需要立即处理 \*/

#define **KERN\_CRIT** KERN\_SOH "2" /\* 严重情况 \*/

#define **KERN\_ERR** KERN\_SOH "3" /\* 错误情况 \*/

#define **KERN\_WARNING** KERN\_SOH "4" /\* 警告情况 \*/

#define **KERN\_NOTICE** KERN\_SOH "5" /\* 注意信息 \*/

#define **KERN\_INFO** KERN\_SOH "6" /\* 普通信息\*/

#define **KERN\_DEBUG** KERN\_SOH "7" /\* 调试信息 \*/

查看当前系统printk打印等级：cat /proc/sys/kernel/printk 4 4 1 7

控制台日志级别： 优先级高于该值的消息将被打印至控制台

默认的消息日志级别： 用该优先级打印没有指定消息级别的消息

最低的控制台日志级别：控制台日志级别可被设置的最小值（最高优先级）

默认的控制台日志级别：控制台日志级别的缺省值

* 一般情况下，优先级高于控制台日志级别的消息将被打印到控制台。优先级低于控制台日志级别的消息将被打印到内核环形缓冲区（dmesg）。

中断

ARM架构中的中断控制器一般叫做GIC

硬中断 软中断

中断上文：响应快，执行时间短，不可被打断

中断下文：耗时，可以被打断

1、tasklet

2、工作队列

向内核申请一个中断

int **request\_irq**(unsigned int irq, //中断号

irq\_handler\_t handler, //中断处理函数

unsigned long flags, //中断标志

const char \* name, //中断名字

void \* dev)

//如果将 flags 设置为 IRQF\_SHARED 的话，dev 用来区分不同的中断，一般情况下将dev 设置为设备结构体，dev 会传递给中断处理函数 irq\_handler\_t 的第二个参数。

flags：include/linux/interrupt.h

IRQF\_SHARED：多个设备共享一个中断线，共享的所有中断都必须指定此标志。如果使用共享中断的话，request\_irq 函数的 dev 参数就是唯一区分他们的标志。

IRQF\_ONESHOT：单次中断，中断执行一次就结束。

IRQF\_TRIGGER\_NONE：无触发。

IRQF\_TRIGGER\_RISING：上升沿触发。

IRQF\_TRIGGER\_FALLING：下降沿触发。

IRQF\_TRIGGER\_HIGH：高电平触发。

IRQF\_TRIGGER\_LOW：低电平触发。

void **free\_irq**(unsigned int irq, void \*dev)

**platform**

平台总线模型

**struct platform\_device** include/linux/platform\_deivce.h

int platform\_device\_register(struct platform\_deivce \*device)

void platform\_device\_unregister(struct platform\_device \*device)

**struct platform\_driver** include/linux/platform\_deivce.h

int platform\_driver\_register(struct platform\_driver \* driver)

void platform\_driver\_unregister(struct platform\_driver \* driver)

设备树

用来描述硬件信息的文件

主要针对ARM架构。

**dts**：device tree source，设备树源码

**dtsi**：device tree source include，设备树源码头文件

**dtc**：device tree compiler，设备树编译器

**dtb**：device tree blob，设备树源码编译后的文件



1、如何编译设备树？

内核源码编译后，在内核源码路径下/scripts/dtc/有一个dtc可执行文件，此为dtc编译器。

编译： **dtc -I dts -O dtb -o xxx.dtb xxx.dts**

反编译：**dtc -I dtb -O dts -o xxx.dts xxx.dtb**

2、设备树语法

标签**:** 设备名称**@**设备地址

如：uart: serial@02288000

同级节点下节点名称不能相同

**/dts-v1/; //**第一行表示dts文件的版本

**/{ //** 根节点 **/**，只能这个名

**uart: serial@02288000{ //** 子节点

**node1\_child{**

**};**

**};**

**led: gpio@22020101{ //** 子节点

**node2\_child{**

**};**

**};**

**};**

**reg属性**

reg = <address1 length1 address2 length2……>;

reg = <0x02200000 0x4000

0x02205000 0x4000

>;

**#address-cells**和**#size-cells**用来描述子节点中的reg信息中的地址和长度信息。

如：

**node1{**

**#address-cells = <1>;**

**#size-cells = <0>;**

**node1\_child{**

**reg = <0>;**

**};**

**};**

**model属性**

如：model = “wm8960-audio”;

model = “this is linux board”;

**compatible属性**

非常重要的属性，用来和驱动进行匹配的，匹配成功之后，会调用驱动中的probe函数

如：compatible = “xvnwei”, “xvnwei-board”;

**status属性**

设备状态，只能为以下值：

okay 设备可用

disabled 设备不可用

fail 设备不可用并检测到了错误

fail-sss 设备不可用并检测到了错误，sss为检测到的错误内容

如：status = “okay”;

**device\_type属性**

只用于描述cpu和memory节点

如：device\_type = “memory”;

·根节点 **/** 下的两个特殊节点：**aliases**和**chosen**

批量定义别名：

**aliases{**

**mmc0 = &sdmmc0;** //第一种写法

**mmc1 = &sdmmc1;**

**mmc2 = &sdhci;**

**serial0 = “/simple@fe000000/serial@11c500”;** //第二种写法

**};**

uboot给内核传递bootargs参数，可以使用chosen节点设置bootargs参数

**chosen{**

**bootargs = “root=/dev/nfs rw nfsroot=192.168.1.1 console=ttyS0,115200”;**

**}；**