## linux命令

ps -ef | grep python 查看指定进程

ifconfig 查询本机的网络信息属性

arp -a 查询同一局域网的所有主机ip

net view 查询局域网主机名称

ln -s (pwd) 创建软链接

mv 移动文件/重命名文件

mknod pwd b/c(字符/块) major minor 创建设备节点

rm -rf /dev/... 不用提示直接删除文件或文件夹

rm -i 逐一询问，-f无需询问，不存在不会提示，-r递归删除

man 函数名 查看c函数的用法

df -hT 查看磁盘空间

更改下载源或网络设置，打开“软件与更新”

sudo update-alternatives --install /usr/bin/python python /usr/bin/python2.7 1

sudo update-alternatives --install /usr/bin/python python /usr/bin/python3.6 2

sudo update-alternatives --config python

更改python版本，update-alternatives符号链接管理工具

ln –s 本文件 链接名

ln –snf 更改本文件 链接名

## Shell相关

chmod +x ./test.sh 使脚本具有执行权限

/bin/sh test.sh 解释器直接运行脚本

expr 表达式计算公式

变量名和等号之间不能有空格

记录时间 `data +%s` 前空格后无空格

source filename shell文件包含

## Driver

bus：总线， paltform： 虚拟总线

return -EFAULT; //errno:14 地址错

return -EAGAIN; //errno:11 资源暂时不可用

return -EINTR; //errno:4 中断的函数调用

return -ESPIPE //errno:29 无效的文件指针重定位

return -ENOTTY; //errno:25 不适当的IO控制操作

重点内容：.owner = THIS\_MODULE为什么加“点”的原因

这种方式称为指定初始化 源自ISO C99标准 初始化不必严格按照定义时的顺序

代码分析：int register\_blkdev(unsigned int major, const char \*name);

major 参数是块设备要使用的主设备号，

name为设备名，它会在/proc/devices中被显示。 如果major为0，

内核会自动分配一个新的主设备号register\_blkdev()函数的返回值就是这个主设备号。如果返回1个负值，表明发生了一个错误。

kmalloc和vmalloc的区别：

1、kmalloc保证分配的内存在物理上是连续的,vmalloc保证的是在虚拟地址空间上的连续

2、kmalloc能分配的大小有限,vmalloc能分配的大小相对较大

3、vmalloc比kmalloc要慢

4、kmallloc使用的是slab内存分配机制，而vmalloc使用的是伙伴系统分配机制，这也是造成它们区别的根本所在

typedef long long loff\_t

typedef unsigned int dev\_t

long(64)/int(32) ssize\_t

unsigned long/unsigned int size\_t

char数组不能设置过大，会出现乱码

inode 文件编号

nd -> i\_rdev 文件所代表的设备号

i\_dev 所在设备的主设备号和此设备号信息

c预处理器（C Preprocessor） CPP, 文本替换工具

预处理指令都是以#开头，在实际编译之前替换成编译器能识别的代码

#define 定义宏 可以模拟函数

预定义宏：编译器事先设定好的宏： \_\_DATE\_\_ \_\_TIME\_\_ \_\_FILE\_\_ \_\_LINE\_\_

错误处理：

erron（整数），perror，strerror, 错误输出文件流：stderr

fprintf(stderr，“”）

perror（“..”）

函数参数数量可变 stdarg.h

动态分配内存：calloc，malloc

char \*description = （char\*）malloc（200 \* sizeof（char））；

释放内存：free

重新分配内存：realloc

命令行参数 int main(int argc, char \*argv[])

argc 参数个数， argv[]指针数组，指向传递给程序的每个参数

没有任何参数时argc是1，argv[0] 存储程序的名称，argv[1] 是一个指向第一个命令行参数的指针

在ram上运行驱动：/usr/src/....

Ubuntu上会产生许多文件:

kernel\_hello.c

kernel\_hello.o

Makefile

kernel\_hello.ko

kernel\_hello.mod

kernel\_hello.mod.c

kernel\_hello.mod.o

Module.symvers

modules.order

RAM上运行驱动位置：/usr/src/linux-headers-$(shell uname -r)

在pc上运行驱动位置：/lib/modules/$(shell uname -r)/build

服务器分为数据服务器和应用服务器

= 是最基本的赋值

:= 是覆盖之前的值

?= 是如果没有被赋值过就赋予等号后面的值

+= 是添加等号后面的值

加@ 在命令行不显示

clear，clean等指令要写在Makefile文件中，之后才能make clear/n

$(MAKE) -C $(KERNELDIR) M = $(CURRENT\_PATH) modules 创建模块ko

$(MAKE) -C $(KERNELDIR) M = $(CURRENT\_PATH) clean 删除模块创建产生的一系列文件

默认执行第一个目标操作 XXX： 相当于语言中的主函数

第一行直接写操作名称即可不用加make，如果换行，则需要用make + 操作名称

例如 caoyong：kernel\_modules

make clear （后面的操作需要加上make)

printk(KERN\_ALERT "Hello, caoyong IoT Studio!\n");

printk函数的参数有0-7八个级别，默认为4

#define KERN\_EMERG "<0>" 系统(不使用) /\* system is unusable \*/

#define KERN\_ALERT "<1>" 立即执行(操作系统使用) /\* action must be taken immediately \*/

#define KERN\_CRIT "<2>" 紧急情况(临界条件) /\* critical conditions \*/

#define KERN\_ERR "<3>" 错误 /\* error conditions\*/

#define KERN\_WARNING "<4>" 警告 /\* warning conditions \*/

#define KERN\_NOTICE "<5>" 提示 /\* normal but significant condition \*/

#define KERN\_INFO "<6>" 正常打印 /\* informational \*/

#define KERN\_DEBUG "<7>" 调试 /\* debug-level messages \*/

在使用设备树时我们将\*\*.dts文件利用dtc编译器编译为\*\*.dtb文件。

在已知\*\*.dtb文件的情况下我们有两种方法可以得到dts源码：

方法一：使用fdtdump工具进行反汇编

使用命令：root#fdtdump \*\*.dtb > temp.dts

反汇编生成的内容保存在temp.dts

方法二：使用dtc编译器进行反汇编

dtc -h 可以产看dtc命令的帮助

使用命令：root# dtc -I dtb -O dts -o temp1.dts s5pv210-smdkv210.dtb

功能 函数原型

打开文件 struct file \*filp\_open(const char \*filename, int flags, int mode)

读取文件 ssize\_t vfs\_read(struct file \*file, char \_\_user \*buf, size\_t count, loff\_t \*pos)

写文件 ssize\_t vfs\_write(struct file \*file, const char \_\_user \*buf, size\_t count, loff\_t \*pos)

关闭文件 int filp\_close(struct file \*filp, fl\_owner\_t id)

pos会随着读写变化

## Makefile语法

.PHONY： 指定伪目标（不管是否存在相同名称文件，该目标一定执行）

make -f rules.txt // make --file=rules.txt 默认执行Makefile文件，当没有时指定txt文件并使用其中的规则

## window命令行 ：

dir 查看当前目录下的文件和文件夹

mdir，md 创建文件夹

echo 向文件中输入

type 查看文件

copy 复制文件

rename move del

## vscode:

ctrl + shift + p 直接调出命令面板

## 代码仓库相关

git pull --rebase origin master 强行更新代码

git checkout --track origin/branch\_name 创建与远端分支相同名称的本地分支

git add .

git commit –v (调出vim编辑器进行提交信息的编辑) -m 直接在命令行输入msg。

**msg示例**：

Headline: A useful human-readable title, up to 72 characters long.

Here is the boday content for the patchset detail descriptions, like function, code fix, side effect,  
test method and so on

Issue: AN-5  
Change-Id: I490fb33ce2197de825e56950c5d07fb0c240d76c

git push origin HEAD:refs/for/metis-main。

git diff filename test.patch //生成patch

git apply test.patch //应用patch

**git commit --amend //修改最后一次提交msg**



repo start 创建并切换分支

git mergetool –tool=bc 使用beyondcompare进行merge

repo forall -c 多仓执行

git rebase 将提交基点改为最新