Linux 编程手册

总纲

本手册的目的是为了规范编程习惯，本文提到的所有规则都是为了保证最终代码的易读、易懂、易维护性等。

若出现文档中未提及的情况，需按照 K&R 风格处理。

第一章　文件结构

第一条 用#include <filename.h>格式来引用标准库的头文件。

第二条 用#include “filename.h”格式来引用非标准库的头文件。

第三条 自定义头文件中，要用#ifndef/#define/#endif 结构产生预处理块防止重复包含，并且与#else 和#endif 对应的宏名要标记完整。

驱动程序及内核态程序如下所示：

```
#ifndef __JZ47XX_H__
#else /* __JZ47XX _H__ */
#endif /* Define __JZ47XX _H__ */
```

应用程序如下所示：

```
#ifndef JZ47XX_H
#else /* JZ47XX _H */
#endif /* Define JZ47XX _H */
```

第四条 头文件中不宜出现类似 extern unsigned int reg_addr 的声明。

第五条 应用程序单个.c 文件长度控制在 500 行左右。

第六条 文件头的格式如下：

```
/*
 * Copyright (c) Ingenic Semiconductor Co., Ltd.
*/
```

第二章　程序的版式

第一条 包含不同级别的头文件要用空行分割。

例如：

```
#include <linux/init.h>
#include <linux/kernel.h>

#include <asm/jzsoc.h>

#include “example.h”
```

第二条 每个函数定义结束之后都要加空行。

第三条 在一个函数体内，逻辑上密切相关的语句之间不加空行，其他地方加空行分隔。控制流程的语句前需要空行，以显得醒目。如break，return，goto等。

第四条 缩进方式为一个 TAB，每个 TAB 占用长度为八个字符，缩进级数不超过三级。

第五条 不要在同一行里放多个语句（包括赋值语句），不要在行尾留空格。

第六条 if、for、while、do 等语句自占一行，执行语句不得紧跟其后。

第七条 关键字 if, switch, case, for, do, while 等之后要加一个空格。函数名之后不要留空格，紧跟左括号‘(’，以与关键字区别。‘(’向后紧跟，‘)’、 ‘,’、 ‘;’向前紧跟，紧跟处不留空格。

在括号表达式内，不要在括号旁加空格。

比如：

```
s = sizeof(struct file);
```

不要写成

```
s = sizeof( struct file ); /* bad */
```

第八条 赋值操作符、比较操作符、算术操作符、逻辑运算符、位域操作符等二元操作符的前后应当加空格。

第九条 一元操作符如“!”、 “~”，“++”、”--“等前后不加空格。

第十条 象“[]“、“.”、“->”这类操作符前后不加空格。

第十一条 代码行最大长度宜控制在 70 至 80 个字符以内。长度超过 80 列的语句可以分成几个有意义的片段，每个片段要明显短于原来的语句，并且放的位置也要明显的靠右。

第十二条 花括号的位置采用 K&R 风格，非函数语句块中（除函数定义外）起始大括号放在行尾，结束大括号放在行首。结束大括号应独自占据一行，除非它后面跟着同一个语句的剩余部分。如果执行体只有一条语句也必须加上{}，方便后来调试程序的人加打印。

第十三条 长表达式要在低优先级操作符处拆分成新行，操作符放在新行之首。

第十四条 代码中需添加适量注释，应尽可能采用 C89 注释风格/* */。注释应当准确、易懂，保证注释和代码的一致性。

第十五条 代码应当在经常会被使用者修改的地方做明显的标记，并且注释密度在此处要尽可能丰富。

第十六条 不要在源代码中包含任何编辑器相关的内容。

第十七条 当定义变量或函数指针时，”*” 应靠近变量或函数名，而非靠近类型名。比如：

```
char *linux_banner;
unsigned long long memparse(char *ptr, char **retptr);
char *match_strdup(substring_t *s);
```

第三章 命名规则

第一条 标识符的长度应当符合“min-length && max-information”原则。

第二条 非 C++程序函数及变量命名应当符合 unix 风格：小写字母、数字、下划线。

第三条 宏或者是常量全用大写字母，用下划线分割单词，宏函数可以用小写字母。

第四条 全局变量和函数应当直观且可以拼读，可望文知义。非外部使用的全局变量应当显

式声明 static。

第五条 局部变量应该言简意赅。

第六条 程序中不能出现仅靠大小写区分的相似的标识符，并且不能出现全大写的变量名。

第四章 表达式和基本语句

第一条 如果代码行中的运算符比较多，用括号确定表达式的操作顺序，避免使用默认的优先级。

第二条 不要编写太复杂的复合表达式，不要有多用途的复合表达式。

第三条 if 语句中，整型变量用“==”或“！=”直接与0 比较；不可将浮点变量用“==”或“！=”与任何数字比较；应当将指针变量用“==”或“！=”与NULL 比较。

第四条 在多重循环中，如果有可能，应当将最长的循环放在最内层，最短的循环放在最外层，以减少CPU 跨切循环层的次数。如果循环体内存在逻辑判断，并且循环次数很大，宜将逻辑判断移到循环体的外面。

第五条 不可在for循环体内修改循环变量，防止for循环失去控制。

第六条 for 语句的循环控制变量的取值采用“半开半闭区间”写法。

For (i = 0; i < 10; i++)

第七条 每个case 语句的结尾不要忘了加break，否则将导致多个分支重叠（除非有意使多个分支重叠）。

第五章 常量

第一条 需要对外公开的常量放在头文件中，不需要对外公开的常量放在定义文件的头部。

第二条 定义几个有关联的常量的时候，应首选枚举类型。使用枚举类型应当显示标明起始值，并且加入前缀与类型，防止未知引用。

例如

```
enum {
        PREFIX_TYPE_EXAMPLE = 0,
};
```

第三条 含有多个语句的宏应该被包含在一个do-while代码块里。

第四条 宏不能影响程序流程，不能包含来源不明的变量。

第五条 应尽量减少使用typedef定义类型。

第六章 函数设计

第一条 参数命名要恰当，顺序要合理。根据平台的ABI多于n个参数，应当传递结构体指针，而通过传入指针的形式获得函数输出结果的参数大于两个时，应当考虑结构体。

第二条 函数的参数书写要完整，不能只写参数的类型而省略参数名字。如果函数没有参数，则用void 填充。函数返回值的类型也同样必须显式声明。

第三条 非外部使用的函数必须显示声明static。

第四条 函数应该短小精悍，并且只实现一个功能，局部变量的数量不应超过5－10个。

第五条 内联函数不宜超过三行。在内联函数与函数宏的优先选择上，如果能够使用内联函数实现的，应当尽量采用内联函数。

第六条 在函数体的“入口处”，对参数的有效性进行检查。定义与执行体不可混淆。

第七条 在函数体的“出口处”，对return语句的正确性和效率进行检查。
第八条 集中函数出口，合理利用goto语句。
第九条 调用具有返回值的函数必须处理其返回值，如果返回值在本层难以处理，须使用断言，或显著的报错，以减少程序掩盖错误，造成调试困难。

第七章 调试信息

第一条 编译过程中每条Warning都必须得到处理。在进行强制数据类型转换时要谨慎。
第二条 代码中应尽量采用分级调试接口，但不能重复定义同类调试接口。尽量避免直接使用printk。对于特定设备的调试信息应使用<linux/device.h>中的宏dev_err(), dev_warn(), dev_info()等，对于不针对某个设备的调试信息，应使用pr_debug() 或pr_info()。
第三条 驱动应当尽量使用内核API，以增强功能的可控性以及内核控制路径的配合程度。
第四条 Ioctl应当使用magic number以及_IOR等宏并独立于驱动程序头文件。
第五条 务必注意内核调试信息的拼写。
第六条 内核调试信息的结尾不需要句号，应明了、无歧义。
第七条 版本发布时应包含言简意赅的调试信息。

第八章 版本控制

第一条 要及时提交代码。
第二条 不允许在同一目录下面设置多个工作目录。
第三条 在本地工作目录下的代码（除当前调试的外）必须是最新的版本或者是基于某个已知版本。
第四条 在提交之前必须要和库里面最新的代码作对比，正确进行提交。