工科：将书边上的所有疑点都用代码实现才算是真的懂！工科知识的转化唯一的方法是代码验证理论。方法：看图说话，图表是最易懂的文字。

# Linux编程思路

1. 与系统打交道的库函数、系统调用一定要做返回值检查
2. 常用的库要理解原理和倒背如流
3. 资源的竞争与释放，调用顺序也会导致竞争
4. Linux为提高效率，会延时写脏页，修改文件时，磁盘存储同步,调用fsync()函数或sync命令
5. 写函数，同一级开关配对、申请内存与释放内存配对、谁new谁delete
6. 用函数时考虑阻塞和非阻塞
7. 释放cpu
8. 资源在多个线程使用要避免竞争，锁一释放，就会存在竞争。
9. 绘制并重审所修改bug所在函数的程序流程图，写代码前写流程图，仔细推敲分支并发情况。
10. 可重入
11. 考虑程序产生的信号，和信号的处理
12. 搜代码：google，github，csdn，pudn，apt-get source code，https://sourceforge.net

技巧：#define sigfillset(ptr) (\*(ptr) = ˜(sigset\_t)0, 0)——宏返回逗号后的值

原子操作思想:atomicoperation:If the operation is performed atomically, either all the s teps are performed, or none are performed.

1. 【变量值】赋值与复位都要有,初值在函数中赋值，不能使用初值，无法保证调用时使用的是初值
2. 线程中调用的子函数中不能含有sleep，否则会有多处阻塞
3. 应用程序中不要出现死循环
4. 写代码时杜绝粘贴、复制
5. 代码架构，分层只是因为上一层数据解析的方式至少有2种以上，即最少有2个子层，要清晰的界定每一层该做的事，层与层之间的接口只留一个全局的结构体即可（简洁）
6. 函数返回前统一修改全局变量，达到出错一个不该，成功全更新的目的，方便管理
7. 分层思想和面对对象的思想并进，降低程序的耦合度和易于理解（较低层次发现错误，在较高层次处理错误），上层定义下层需要提供的接口（方法&属性）
8. 有交互的就放一个线程处理，没有交互的就放多个线程处理
9. 对接口（全局变量）应在同一个地方同时更新，避免只更新了一部分，导致程序状态错乱

**高质量c语言**

**原则：易于理解、便于维护**

1. 对数据操作避免裸数据，应使用结构体来定义数据，及操作——优点：改动时，只要改动结构体就行了，改一处即可，不必处处都更改。

2.尽量少使用字符串数组来作为描述——优点：对应关系明显，多处调用时得多处修改，易于维护。

3.枚举值统一用大写。

**原则：**避免命名冲突

1.自定义的数据结构统一为全大写，全局变量增加g\_xxx(xxx为大写源文件名)前缀，静态变量增加s\_前缀，避免命名冲突，结构体成员加前缀，如：

typedef struct os\_flag\_grp

{ /\* Event Flag Group \*/

INT8U u8OSFlagType; /\* Should be set to OS\_EVENT\_TYPE\_FLAG\*/

void \*pOSFlagWaitList; /\* Pointer to first NODE of task waiting on event flag\*/

OS\_FLAGS OSFlagFlags; /\* 8, 16 or 32 bit flags \*/

#if OS\_FLAG\_NAME\_EN > 0u

INT8U \*pOSFlagName;

#endif

} OS\_FLAG\_GRP;

Int g\_iFTPIp;

Static int s\_iIp;

2.文件命名统一小写 xxx\_xxx.h或xxx.h，Windows系统不区分大小写，但是Linux系统则区分，如：stm32f4xx\_spi.c ，ftp.h

3.函数命名:接口采用：xxx\_主\_胃、xxx\_动词或xxx\_动词\_名词；非接口采用：static主\_胃、static xxx\_动词或static xxx\_动词\_名词;形参为指针使用const修饰

**原则：**质量优先

0. 可读性，易于理解第一位。

1. 正确性，指程序要实现设计要求的功能

2. 简洁性，指程序易于理解并且易于实现——**逻辑上优先处理异常**

3. 可维护性，指程序被修改的能力，包括纠错、改进、新需求或功能规格变化的适应能力

4. 可靠性，指程序在给定时间间隔和环境条件下，按设计要求成功运行程序的概率。

优化：1.结构体储存结构 2.if swtich分支按触发概率来排序书写，且循环时不出现函数调用

3.避免反复从内存加载数据到cache，如：多维数据，回跳跃式访问数组成员

4. 创建资源库，以减少分配对象的开销，如：队列采用数组来实现。

**接口设计原则：**隐藏细节

同一级别创建与释放资源；较低层次发现错误，在较高层次处理错误；减少耦合；参数取值校验；参数的属性

程序调试：依照别人的51程序，或者是给的相应的程序，明白每一个程序的功能，和要修改的地方，做好记录。程序出错时，先用单步调试，看自己定义的每一个变量的值，明白程序的数据流动变化过程，并结合测试工具（示波器，万用表，频谱分析仪等）对程序进行分析检查；最后进行压力测试

程序心情的调整：

a.调程序，明白最小单元，结合分析最小单元的实现

b.写好程序的流程图和调试时按流程调试

c.看状态寄存器的值，明白程序的状态

d.如何选择合适的时机与别人交流互动：团队精神重要，在程序分析不出问题时——1.说明文档 2.帮助手册 3.网页和论坛 4.中国知网 5.技术支持6.书籍7.同事 8.FQA中

程序良好的习惯

1. 变量命名，及数量类型的选取，多局部变量少全局变量；
2. 程序计算是的注解及精度（四舍五入，先乘后除）；
3. 程序注解与代码相对应，结构清晰，各个不同功能语句分割；
4. 软件测试和硬件测试，符合规定指标，并考虑多种应用情况；
5. 严格按照协议要求，防患未然；
6. 每个功能函数都要测试是否能实现，整体流程是否正确；
7. 程序流程设计及功能语句多请教别人，虚心请教多多交流；
8. 程序调试时，按照思想流程设计无误后，若无法实现，清楚明白问题所在，团队解决；
9. 技术文档参数多问如何实现；
10. 写程序做出成果，多想想衡量好坏的标准，凡事都有一个好坏标准。
11. 代码可扩展性
12. 变量名尽可能短，过长可能会导致编译器出错，类型转换和有符号变无符号，如keil4

学习方法心得：

1.通读理论：a.对知识点进行全面扫盲，知道有这么一回事，对以后定位问题(背目录)大有帮助，工科的最可悲在于无知；b.记录哪些是常用的技术，哪些是过时的技术，在代码验证时掌握最常用的就行了，其所替代的知识就不用深究，反正也不用；c.思考问题的大致方式 d.查资料、求助的途径 e.如何利用开源代码

2.代码验证：i.先运行代码，对通读的疑问点进行解惑（代码验证书本的内容）；ii.明白基本功能代码(可移植的单元代码)，类似I/O，中断，串口等方便重组；iii.熟悉项目编写思维方式、Debug方式、原则。iV.明白该功能到底是个怎么个效果 V.记录哪些是可以继承前人的成果。

3.实践项目：1）站在巨人的肩膀上（向业内人士进行咨询，向书本寻找相关的demo）2）增量建模技巧，先破0（平台搭建，功能开发），再精细到业务需求。

ios第三方库使用心得：

快速了解第三方库：1.运行库里的demo并结合github的README.md或是官网资料介绍理解；2.通过中断和中断的输出信息，大致定位程序流程，弄明白库里的对象、对象方法（初始化用）、数据源协议（数据显示用）、协议（触发的动作，对外的输出）3.有问题在issue里面先找，然后在百度，解决不了就发布issue 4.实践之前多看看别人是怎么做的，有没有利用开源第三方库。

C语言TCP/IP协议移植心得：协议的数据封装和解封装、断点跟踪程序进入的函数和全局变量（C程序的接口）。

USB协议代码实现心得：协议的分层是因为上一层数据解析的方式至少有2种以上，（物理层和数据链路层用来得到package数据，必须要有）根据协议的数据流图（对数据位的不同解析）来实现。

Ucos-II移植ftp心得

1.ucos-II提供的是一个库，最小内核包括时间管理、多线程管理、内存管理、线程通信构成内核，内核的扩展包括一种/多种网络管理、一种/多种文件系统管理再集中抽象 对外提供统一接口，硬件驱动对上统一抽象接口为文件，构成复杂的linux内核