##### 设备oom定位方法总结

Author：zhujiajia

总体思路是 分析是否泄漏——审查代码——工具定位

###### 一、分析是否泄漏

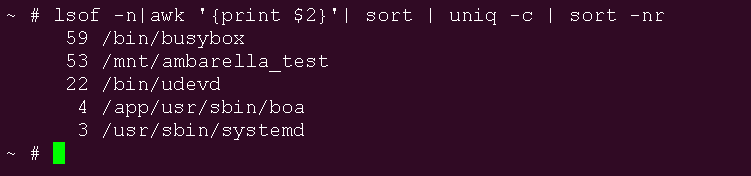
在嵌入式系统中，内存泄漏一般为文件句柄泄漏、堆内存泄漏

1. 文件句柄泄漏

如打开的文件、socket、proc等。Linux一切皆文件的思想，系统很多资源都是文件的概念，open一个资源不去close，就会造成句柄泄漏。

Linux lsof命令可以查看进程的句柄数目

lsof -n|awk '{print $2}'| sort | uniq -c | sort -nr



观察一段时间，如果句柄有泄漏，进程的句柄数会持续上升。

1. 堆内存泄漏

malloc/free,new/delete,必须成对出现，在堆上分配内存不释放，则会造成堆内存泄漏

Linux top/free 可以查看系统内存的信息

**~ #Free**

系统可用内存 = free+buffers+cached，如果可用内存持续下降，则存在oom的情况

Buffer 主要是磁盘的缓存

Cached 主要是内核的page页及一些数据暂存,cached用来提高效率

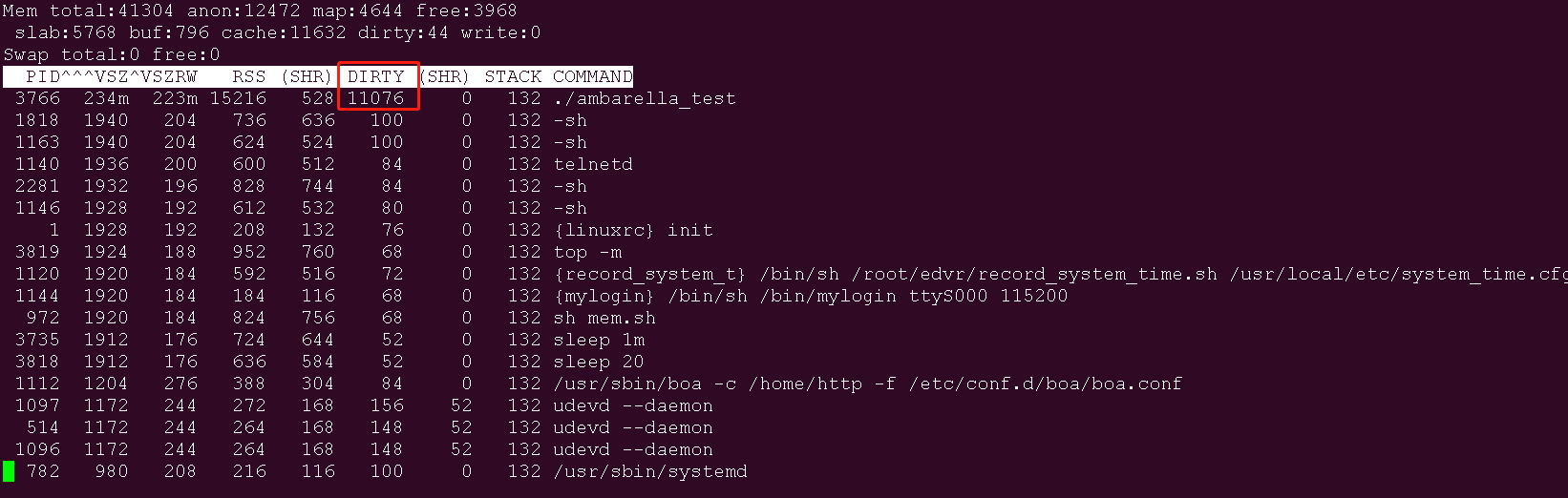
Shared 共享内存，共享内存操作不当也会造成oom,若该值持续增大，则存在oom的情况



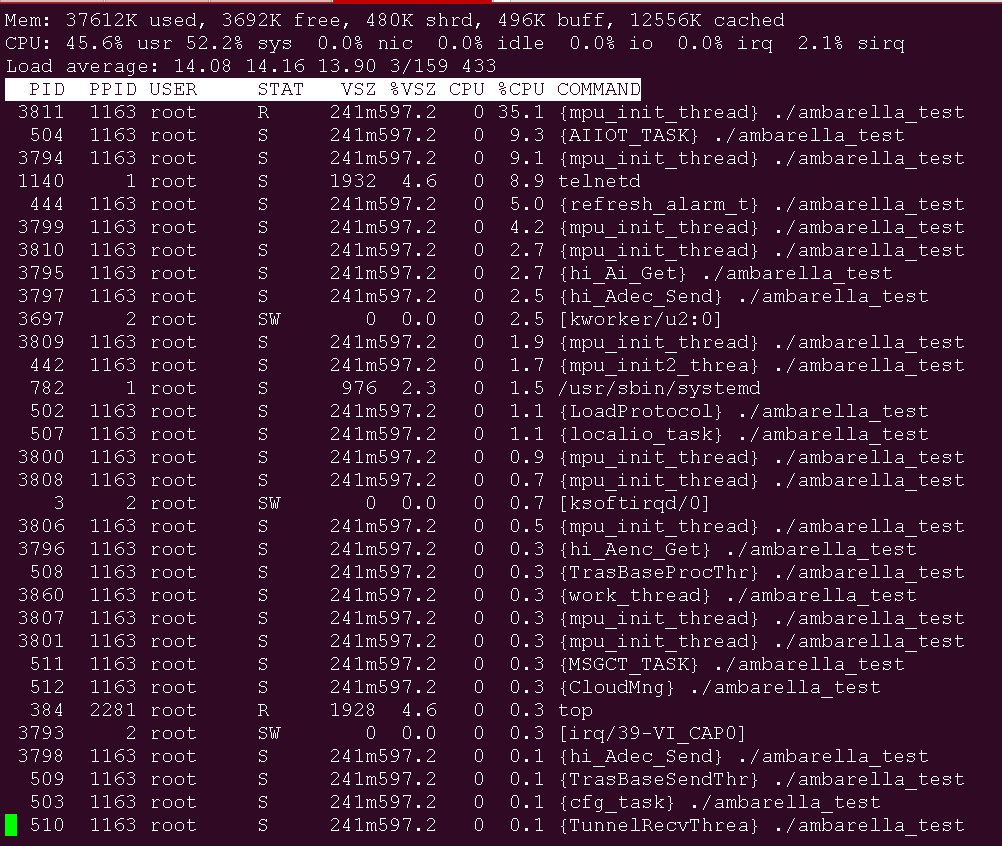
**~ #Top -m**

如果DIRTY(脏内存)持续增大，则存在oom的情况

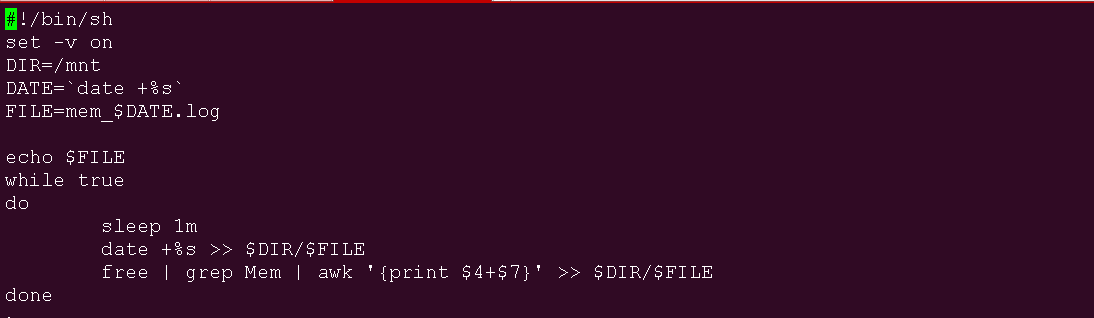
top -m -b -n 10000 -d 3 >> /yourfilepath 可使用top动态打印到文件中



* 在top界面 Shift+H, 可以查看系统中线程信息。%CPU可以表线程是否死循环。



* 使用系统命令配合脚本工具，则可以收集内存运行的数据，方便问题定位。数据配合python画图更直观举例，1min统计一下free+cached值



###### 二、代码审查

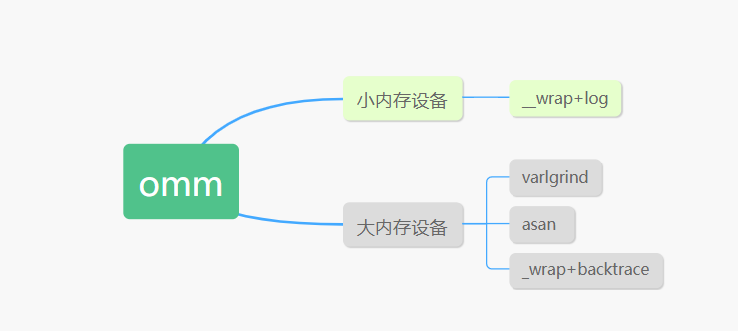
大致了解泄漏问题后，可以先进行代码审查，一般情况是新增的功能引入，对应进行代码review，审查可配合环境测试，缩小业务范围确定哪个模块泄漏。

代码上个人觉得一般常见于下面情况：

1. **内存泄露** 
   1. 多个分支跳出漏释放
   2. 线程异步释放阻塞或遍历不完全
   3. 一些包含malloc的接口，如JSON Create或Parse后未delete
   4. 涉及字符串系统调用函数多检查，是否存在越界
2. **句柄泄露**
   1. 多个分支跳出时句柄忘关

###### **三、工具使用**

如果上面还无法定位，则需要使用工具定位了，工具对编译器、系统内存都有要求，嵌入式设备上限制有很多。

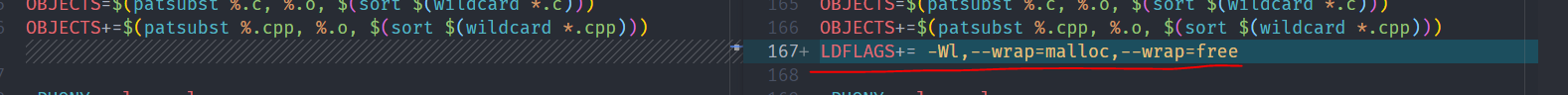


**oom定位工具**

* Varlgrind(实际项目使用比较坑，吃设备性能、回溯要求比较多，且对系统运行速度影响较大) <https://blog.csdn.net/ljq32/article/details/51479896>
* ASAN(集成在GCC中,对系统性能要求比Varlgrind要低,比较轻量,未使用过) <https://blog.csdn.net/wads23456/article/details/104836179/>
* Backtrace是linux接口，可以打印当前堆栈信息，Backtrace对编译器有要求，backtrace配合signal还可以抛出段错误时的堆栈，来定位段错误问题，arm-himix100-gcc不支持backtrace函数； <https://blog.csdn.net/gongmin856/article/details/79192259>(使用backtrace定位段错误问题)
* 使用\_\_wrap+backtrace可以抛出堆内存分配异常处的堆栈信息，可以定位到具体的代码行数，可参考如下:<https://zhuanlan.zhihu.com/p/158370711>

若工具不能使用，可以使用\_\_wrap。步骤如下：

1. 修改makefile，添加链接选项 LDFLAGS+= -Wl,--wrap=malloc,--wrap=free，涉及到的库都需要增加该链接选项



1. 定义malloc/free，修改见比较报告(代码上还可以做一下统计)



1. 编译后运行，可以看到打印信息，通过size和ptr分析是否存在内存未释放的位置

