全志科技
Allwinner Technology

Y10

内存配置说明

文档履历

版本号	日期	制/修订人	制/修订记录
V1.0	2014-07-30		初始版本
V2.0	2014-09-02		增加 CMA 配置说明, 修改 ION 预留大小说明.

目　　录

概述

1.1. 编写目的

介绍平板方案内存配置说明相关知识, 供方案定制和开发人员参考。

1.2. 适用范围

适用于 Y10 平台；

1.3. 相关人员

本文档的参考人员为 Y10 方案定制或开发人员.。

2. 术语、缩略语及概念

2.1. 术语、定义、缩略语

2.1.1. 预留内存

linux 标准函数不能分配超过 4M 的连续物理内存, 而硬件模块有时需要大于 4M 的连续物理内存. 预留内存就是为了解决这个问题.

2.1.2. vmalloc 区

指 linux 内核虚拟地址空间中, 0xFF000000 之前的一段区间, 大小不能超过 976M;

这段区间用于物理内存的动态映射, io 虚拟地址, vamlloc 函数等. 与它相对应的是低端内存区, 即线性映射区;

2.1.3. ION

android 引入的内存管理框架, 在 kernel 实现, 主要用于应用层访问连续物理内存.

2.1.4. zram

即压缩内存机制. 在系统内存紧张时, 将不活动内存进行压缩, 并回写到一块压缩内存区域, 以提高内存利用率.

2.1.5. CMA

连续内存分配器, Continuous Memory Allocator, 从 linux-3.5 引入.

CMA 实现了预留内存的充分利用. 通过 CMA, 预留内存的空闲部分可以被其他模块利用, 通过 alloc_page 申请, 从而避免了浪费.

2.2. 概念阐述

2.2.1. drop_cache 机制

指定期对文件系统缓存进行回收的机制. linux 会将空闲内存大量用于 fs 缓存, 这部分内存若不手动回收, 会导致系统空闲内存紧张, 影响效率.

android 原生不会 drop_cache, 我们在 framework 中添加了 drop_cache 机制, 每隔几秒回收一次.

哪些情况才需要回收? 根据系统总可见内存, 当前空闲内存, fs 缓存的大小和比例来定.

2.2.2. lowmemorykiller 机制

指在系统空闲内存较少时, 杀死重要性低的应用, 回收内存的机制.

系统定义了若干个进程优先级, 比如前台应用优先级为 0, 后台应用优先级为 9~15, 数值越低, 优先级越大.

在每次进行内存回收时, 选取一个优先级最低, 且占用内存最多的应用杀掉.

内存回收的时机如何定? 根据系统总可见内存, 当前空闲内存和 fs 缓存大小来定.

3. kernel 相关配置

3.1. 通用内核配置

512M 和 1G 通用的内核 menuconfig 配置.

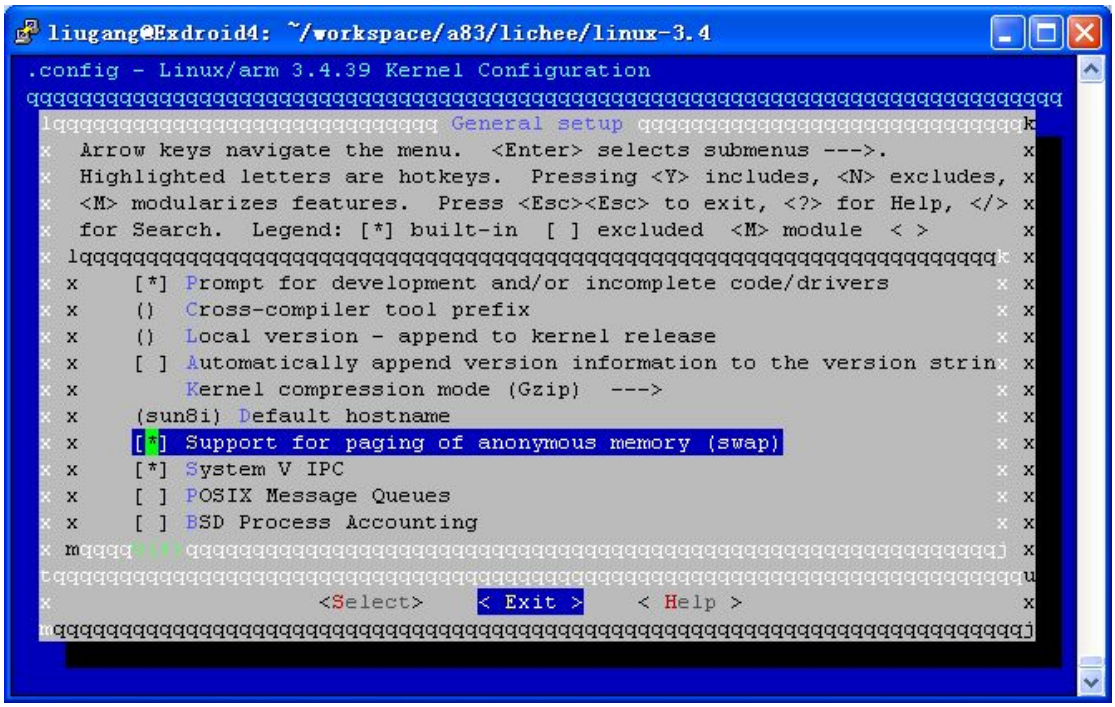
3.1.1. zram 配置

(1) CONFIG_SWAP:

1 在 linux3.4 目录下，输入 make ARCH=arm menuconfig

2 按照以下选项依次选择：

General setup ---> Support for paging of anonymous memory (swap)



(2) CONFIG_ZRAM

ZRAM 设置方法：

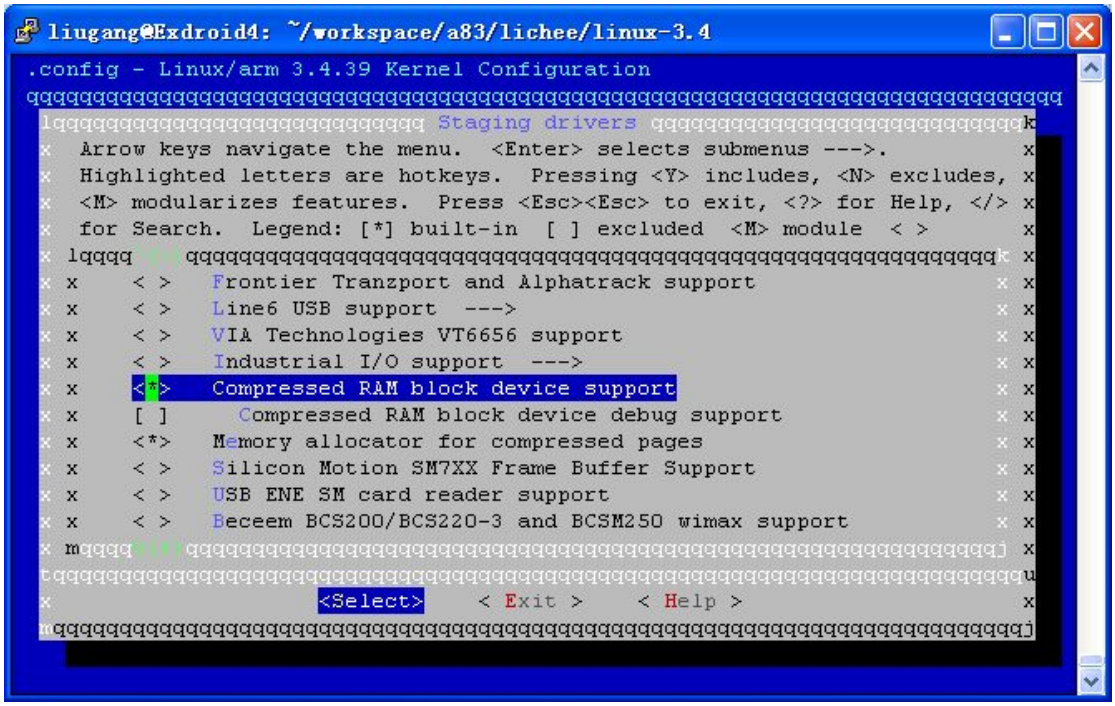
1 在 linux3.4 目录下，输入 make ARCH=arm menuconfig

2 按照以下选项依次选择：

Device Drivers ---> Staging drivers

---> Compressed RAM block device support

---> Memory allocator for compressed pages



3.1.2. lowmemorykiller 配置

lowmemorykiller 设置方法：

1 在 linux3.4 目录下，输入 make ARCH=arm menuconfig

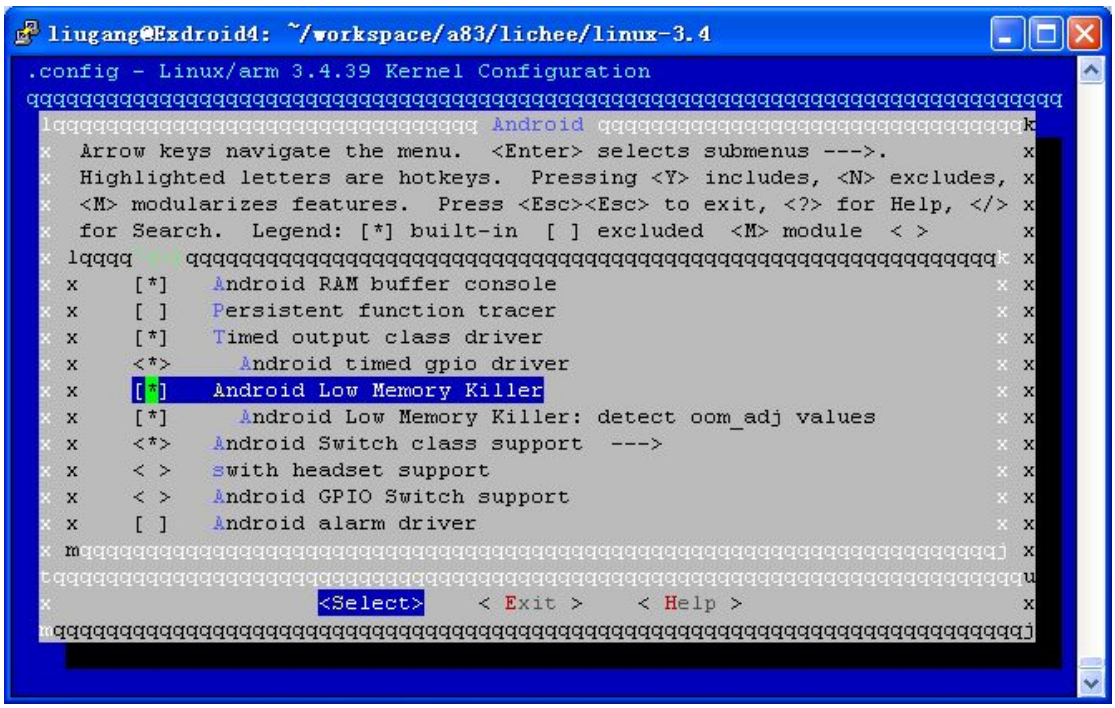
2 按照以下选项依次选择：

Device Drivers ---> Staging drivers

---> Compressed RAM block device support -> Android

---> Android Low Memory Killer

---> Android Low Memory Killer: detect oom_adj values

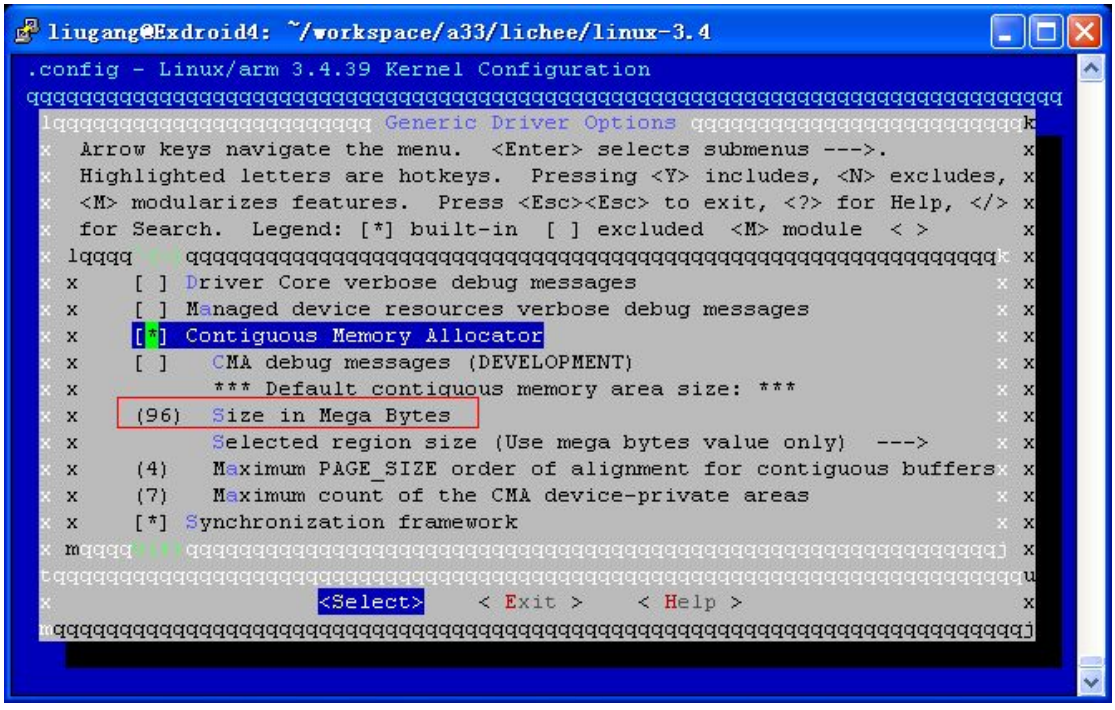


3.1.3. drop_caches 节点写使能

```
diff --git a/kernel/sysctl.c b/kernel/sysctl.c
index 49f47258..48ba6b0 100644
--- a/kernel/sysctl.c
+++ b/kernel/sysctl.c
@@ -1167,7 +1167,7 @@ static struct ctl_table vm_table[] = {
                .procname       = "drop_caches",
                .data           = &sysctl_drop_caches,
                .maxlen         = sizeof(int),
-               .mode           = 0644,
+               .mode           = 0666,
                .proc_handler   = drop_caches_sysctl_handler,
                .extra1         = &one,
                .extra2         = &three,
```

3.1.4. CMA 配置

(1) Device Drivers ---> Generic Driver Options ---> Contiguous Memory Allocator

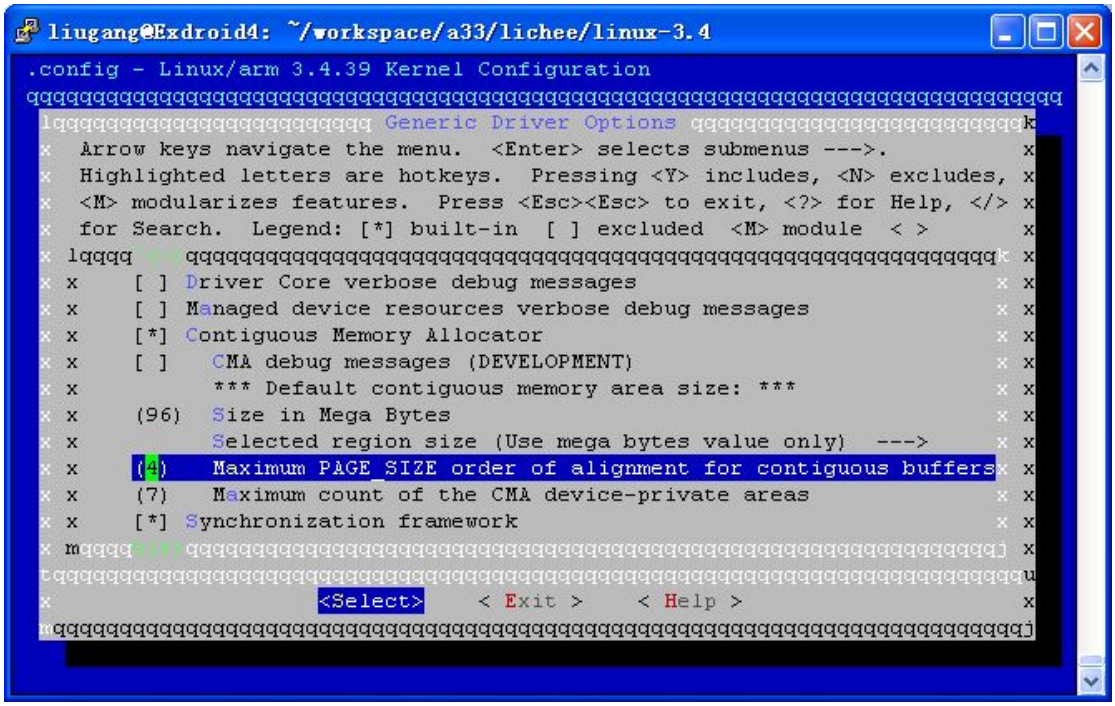


注: 上述"Size in Mega Bytes"的配置不起作用, 保持默认即可.

(2) "Maximum PAGE_SIZE order of alignment for contiguous buffers"

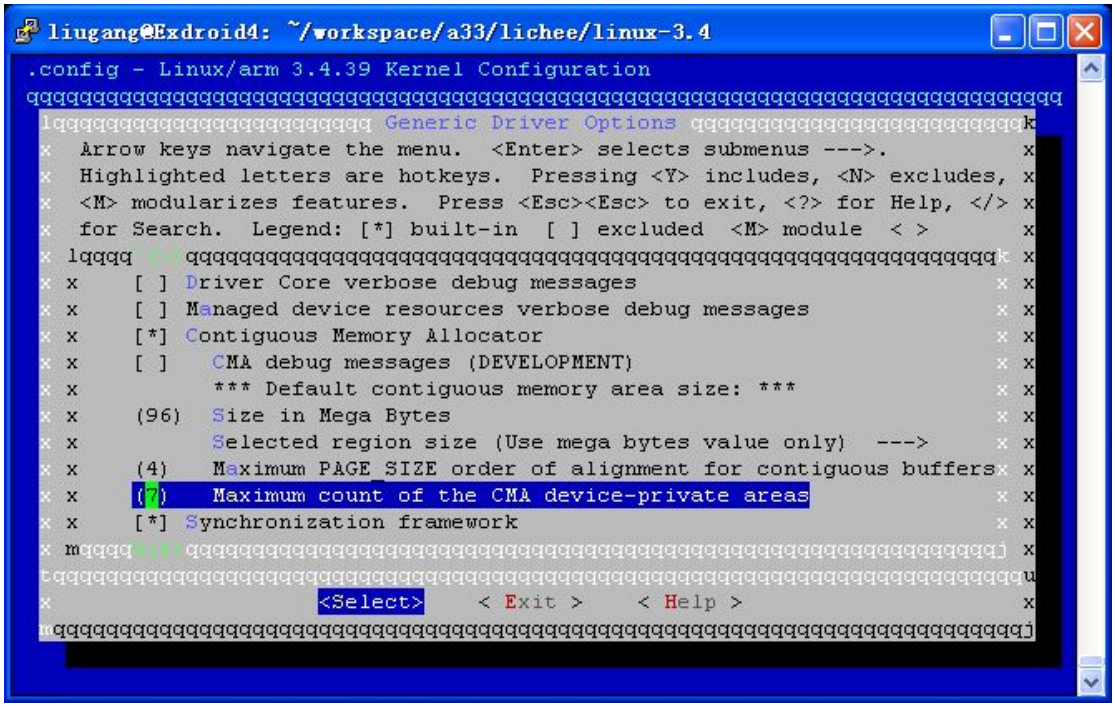
描述 CMA 分配内存时, 起始地址的对齐大小. 这里为 4, 表示每次分配时, 起始地址按 2^4 个 PAGE_SIZE (即 64K) 对齐.

保持默认即可, 无须修改.



(3) “Maximum count of the CMA device-private areas”

描述最多支持的设备私有 CMA 空间的个数. 目前我们用的是系统 CMA 区间, 没有用到设备私有区间, 因此该配置项无用.



3.2. ION 预留区大小

ION 预留多少合适? 需根据实际需要来定, 不同方案不一样, 估值依据如下:

1. 几个主要模块的内存消耗: GPU, VE, CAMERA, DISPLAY;
2. 规格场景的内存消耗: mirecast, 3D 游戏, CTS/GMS;

预留大小如何确定？一般分两步:

1. 根据规格，估算主要场景下，GPU/VE/CAMERA 等各模块消耗的内存，计算出总和 total;
2. 采用试凑法，将预留大小设为 total，测试主要场景下，是否有 ION 申请失败的打印.
3. 若有申请失败情况，则逐渐加大预留量，比如每次增加 16M，直到所有场景压力测试通过为止.
4. 内核配置了 CONFIG_CMA 时，预留内存会（比不使用 CMA）适当加大，以降低 ION 分配失败的概率.

ION 预留内存大小设置方法：

统一在 lichee/tools/pack/chips/sun8iw5p1/configs/default/env.cfg 中设置:

```
ion_cma_512m=112m ion_cma_1g=176m ion_carveout_512m=96m ion_carveout_1g=150m
```

代码会根据是否打开了 CONFIG_CMA，以及总内存大小，来决定选取上述哪个作为最终预留大小.

3.2.1. 512M 方案

当未选择 CONFIG_CMA 时，由 ion_carveout_512m 决定 ION 预留内存大小. 按上述配置，此时预留大小为 96 MBytes.

当选择了 CONFIG_CMA 时，由 ion_cma_512m 决定 ION 预留内存大小. 按上述配置，此时预留大小为 112 MBytes.

3.2.2. 1G 方案

当未选择 CONFIG_CMA 时，由 ion_carveout_1g 决定 ION 预留内存大小. 按上述配置，此时预留大小为 150 MBytes.

当选择了 CONFIG_CMA 时，由 ion_cma_1g 决定 ION 预留内存大小. 按上述配置，此时预留大小为 176 MBytes.

3.3. vmalloc 区大小

3.3.1. 512M 方案

512M 方案下, vmalloc 区默认大小为 496M，从 0xD0000000 到 0xFF000000;

512M 方案使用默认配置即可，无须改动.

3.3.2. 1G 方案

1G 及以上方案中, vmalloc 区默认大小为 248M，从 0xEF800000 到 0xFF000000;

在 ION 内存消耗较大的场景下，比如 miracast/3D 游戏/GMS/CTS，默认 248M 可能不能满足需求，因此建议将 vmalloc 区增大到 384M.

方法是在命令行增加”vmalloc=384m”，以下方式任选一:

1. 在方案 env.cfg 文件中，增加”vmalloc=384m”信息:

lichee\tools\pack\chips\sun8iw5p1\configs\default\env.cfg:

```
setargs_nand=setenv bootargs console=${console} root=${nand_root} vmalloc=384M
```

```
init=${init} loglevel=${loglevel} partitions=${partitions}
setargs_mmc=setenv bootargs console=${console} root=${mmc_root} vmalloc=384M
init=${init} loglevel=${loglevel} partitions=${partitions}
```

上述 sun8iw5p1 对应 Y10.

2. 在方案中增加”vmalloc=384m”:

android\device\softwinner\airforce-evb\BoardConfig.mk:

```
BOARD_KERNEL_CMDLINE += vmalloc=384M
```

将上述 airforce-evb 替换为实际方案目录.

4. android 相关配置

4.1. drop_cache 门限

2.1.1. 512M 方案

文件: android\frameworks\base\services\java\com\android\server\am\
ActivityManagerService.java

```
        private boolean flushCache(int nCache, int nFree){
            boolean nRet = true;
            if((nFree < 20000 && nCache > 70000) || (nFree < 10000 && nCache > 30000)){
                nRet = writeFile("/proc/sys/vm/drop_caches", "3");
            }
            if((nCache - nFree) > 80000) {
                nRet = writeFile("/proc/sys/vm/drop_caches", "1");
            }
            return nRet;
        }
```

上述 nFree, nCache 分别表示空闲内存和 fs 缓存, 单位为 KByte;

2.1.2. 1G 方案

1G 方案目前没有使能 drop_cache 机制, 不需要配置.

注: 上述 flushCache 函数仅当 sys.mem.opt 为 true 才会执行, sys.mem.opt 在 system/core/init/property_service.c 里面配置:

```
    if (get_dram_size() > 512) {
        property_set("dalvik.vm.heapsize", "384m");
        property_set("dalvik.vm.heapstartsize", "8m");
        property_set("dalvik.vm.heapgrowthlimit", "96m");
        property_set("dalvik.vm.heapminfree", "2m");
        property_set("dalvik.vm.heapmaxfree", "8m");
        property_set("sys.mem.opt", "false");
        property_set("ro.config.low_ram", "false");
    } else {
        property_set("dalvik.vm.heapsize", "184m");
        property_set("dalvik.vm.heapstartsize", "5m");
        property_set("dalvik.vm.heapgrowthlimit", "48m");
        property_set("dalvik.vm.heapminfree", "512K");
        property_set("dalvik.vm.heapmaxfree", "2m");
        //aw use
        if(strcmp(buf,"true")){
            property_set("sys.mem.opt", "true");
```

```
        }
        property_set("ro.config.low_ram", "true");
    }
```

根据上述代码，大于 512M 的方案 sys.mem.opt 被设为 false，因此 flushCache 不起作用.

4.2. lowmemorykiller 门限

2.1.3. 512M 方案

512M 方案运行大型游戏，或 3D 跑分软件时，会出现内存不足. 因此需要适当调整 lmk 门限，以便及时杀后台进程，释放内存.

文件: android\frameworks\base\services\java\com\android\server\am\ProcessList.java

```
        if (mTotalMemMb > 256 && mTotalMemMb <= 512) {
            memString.delete(0,memString.length());
            memString.append("4096,6144,8192,10240,12288,14336");
            mOomMinFree[0] = 16384; //  即 4096 * 4
            mOomMinFree[1] = 24576; //  即 6144 * 4
            mOomMinFree[2] = 32768; //  即 8192 * 4
            mOomMinFree[3] = 40960; //  即 10240 * 4
            mOomMinFree[4] = 49152; //  即 12288 * 4
            mOomMinFree[5] = 57344; //  即 14336 * 4

            adjString.delete(0,adjString.length());
            adjString.append("0,1,2,4,9,15");
        }
```

上述配置的意思是:

(1) 当 MemFree 和 Cached 的内存 (通过 cat /proc/meminfo 查看) 少于 14336*4K 时，优先级在 15 以上的后台进程会被杀;

(2) 当 MemFree 和 Cached 的内存少于 12288*4K 时，优先级在 9 以上的后台进程会被杀;

(3) 当 MemFree 和 Cached 的内存少于 10240*4K 时，优先级在 4 以上的后台进程会被杀;

(4) 当 MemFree 和 Cached 的内存少于 8192*4K 时，优先级在 2 以上的后台进程会被杀;

(5) 当 MemFree 和 Cached 的内存少于 6144*4K 时，优先级在 1 以上的后台进程会被杀;

(6) 当 MemFree 和 Cached 的内存少于 4096*4K 时，优先级在 0 以上的进程会被杀; (此时前台进程有可能被杀，因为其优先级为 0)

2.1.4. 1G 方案

使用默认配置，暂不调整.

2.2. zram disksize

2.2.1. 512M 方案

512M 方案上, zram disksize 设为 384M.

文件: android\device\softwinner\airforce-evb\fstab.sun8i

/dev/block/zram0 none swap defaults zramsize=402653184

2.2.2. 1G 方案

1G 方案上, zram disksize 设为 512M.

文件: android\device\softwinner\airforce-evb\fstab.sun8i
/dev/block/zram0 none swap defaults zramsize=536870912

4.3. dalvik heap 参数

dalvik heap 参数会限制进程分配的内存大小, 在总内存很多时, 值越大则应用响应越快; 但在总内存较少时, 必须限制该值, 以免应用占用过多内存, 导致内核运行紧张.

2.2.3. 512M 方案

文件: android\system\core\init\property_service.c

```
static int enable_adaptive_memory(void)
{
...
    //for memory > 1024,
    if (get_dram_size() > 512) {
        property_set("dalvik.vm.heapsize", "384m");
        property_set("dalvik.vm.heapstartsize", "8m");
        property_set("dalvik.vm.heapgrowthlimit", "96m");
        property_set("dalvik.vm.heapminfree", "2m");
        property_set("dalvik.vm.heapmaxfree", "8m");
        property_set("sys.mem.opt", "false");
        property_set("ro.config.low_ram", "false");
    } else {    ...
```

2.2.4. 1G 方案

文件: android\system\core\init\property_service.c

```
static int enable_adaptive_memory(void)
{
...
    //for memory > 1024,
    if (get_dram_size() > 512) {
...
    } else {
        property_set("dalvik.vm.heapsize", "184m");
        property_set("dalvik.vm.heapstartsize", "5m");
        property_set("dalvik.vm.heapgrowthlimit", "48m");
        property_set("dalvik.vm.heapminfree", "512K");
        property_set("dalvik.vm.heapmaxfree", "2m");
        //aw use
        if(strcmp(buf,"true")){
```

```
                property_set("sys.mem.opt", "true");
            }
            property_set("ro.config.low_ram", "true");
        }
        return 0;
}
```