

KT 密级状态: 绝密() 秘密() 内部(☒) 公开()

Android malloc_debug 调试机制索引

(图形显示平台中心)

文件状态: [<input checked="" type="checkbox"/>] 正在修改 [<input type="checkbox"/>] 正式发布	当前版本:	V0.4
	作 者:	陈真, 李煌
	完成日期:	2019/7/12
	审 核:	
	完成日期:	

瑞芯微电子股份有限公司

Fuzhou Rockchips Semiconductor Co., Ltd

(版本所有, 翻版必究)

版 本 历 史

版本号	作者	修改日期	修改说明	备注
V0.1	陈真	2017/5/25	初稿	
V0.2	李煌	2019/1/31	增加命令行方式，增加 skqp 案例	
V0.3	李煌	2019/2/14	增加属性设置参数的说明 与介绍	
V0.4	李煌	2019/7/12	增加对 api 方式的说明和 修改	

目录

Android malloc_debug 调试机制索引.....	1
1 概述.....	4
2 官方文档索引和补充说明.....	4
2.1 属性设置参数说明.....	5
2.2 具体使用说明.....	5
3 在 surfaceflinger 进程中定位内存泄漏的例子	7
3.1 用例和问题现象.....	7
3.2 调试策略.....	8
3.3 实现获取 allocation_info 的 patch.....	8
3.4 获取 dump_files_of_allocation_info 的具体步骤	8
3.5 对 dump_files_of_allocation_info 的比较分析	9
4 Cts skqp 测试内存泄漏的例子	10
4.1 用例和问题现象.....	10
4.2 调试步骤.....	10
4.3 对 heap_info.txt 的比较分析	11

1 概述

本文提供关于 `malloc_debug` 资料和文档的 索引信息, 并提供一个在 `Android 7.1` 中, 调查 `surfaceflinger` 中内存泄漏的简单例子和 `patch`.

`malloc_debug` 可用于 调试 发生在 "来自 `system heap` 的 `memory block`" 上的 泄漏, 访问越界等问题.

该工具通过属性来使能, 当使能 `malloc debug` 时, 该工具会替换以下 `alloc` 的调用 `api`.

- `malloc`
- `free`
- `calloc`
- `realloc`
- `posix_memalign`
- `memalign`
- `aligned_alloc`
- `malloc_usable_size`
- `pvalloc`
- `valloc`

2 官方文档索引和补充说明

`malloc_debug` 的官方文档主要有如下两份 :

https://android.googlesource.com/platform/bionic/+/master/libc/malloc_debug/README.md

https://android.googlesource.com/platform/bionic/+/master/libc/malloc_debug/README_api.md

使用前, 请仔细阅读.

2.1 属性设置参数说明

该工具通过属性设置，这里对不同的属性参数进行说明。仅对常用的进行说明。

backtrace[=MAX_FRAMES]

使能来获取 alloc 的堆栈。系统运行会变得缓慢，如果系统配置该属性后过于卡顿可以减小 MAX_FRAMES 的值。MAX_FRAMES 未指定的话，默认值为 16。

leak_track

使能后可以打印出进程释放后仍然存在的内存泄漏，可以用于排查进程推出后内存仍未释放的情况。

record_allocs[=TOTAL_ENTRIES]

记录所有 alloc，free 动作并保存为文件。

该属性 Android O 之后才支持

2.2 具体使用说明

简要归纳总结两种 debug 内存泄漏方式，仅 Android 7.1 及以上可以使用。

(1) 命令行方式: 适用于应用进程

输入以下命令打开 malloc_debug，可以仅指定 app_process 即对应 app 的进程名，如果没有该属性，默认全局追踪的 malloc_heap 的信息。

```
adb shell stop
adb shell setprop libc.debug.malloc.options backtrace
adb shell start
```

需要将特定 app 的 malloc_heap 信息保存下来, <PID_TO_DUMP> 为 app 的 pid, 通过 busybox ps 查看。am 命令只能抓取应用进程。

```
adb shell am dumpheap -n <PID_TO_DUMP> /data/local/tmp/heap.txt
```

但是 heap 信息并不能直接查看, 需要 google 的一个 native_heap_viewer.py 的工具进行转换。
该工具位于 sdk 中以下目录, 7.1 sdk 可能没有该 py 脚本, 需要从 8.1 工程拷贝过来。

```
development/scripts/native_heapdump_viewer.py
```

通过以下命令将 heap.txt 进行转换, 可以获取到堆栈信息。

```
python development/scripts/native_heapdump_viewer.py --symbols  
</some/path/to/symbols/> heap.txt > heap_info.txt
```

注: --symbols 需要指定为 out 目录下对应目录的 symbols 目录, 否则获取不到堆栈信息。

native_heap_viewer.py 的工具也支持输出为 html, 可以更方便的在网页上进行查看, 只要加入 --html 选项即可。

比如:

```
python development/scripts/native_heapdump_viewer.py --html --symbols  
</some/path/to/symbols/> heap.txt > heap_info.html
```

html 和 txt 文件的方式各有优点, txt 文件需要间隔一定时间获取两份 heap_info.txt 用 beyond compare 或者类似比对软件比对前后两份 alloc 数据量, 来确认哪个函数调用存在内存泄漏。Html 的方式可以方便折叠堆栈, 但是需要人眼手工比对。

(2) api 方式: 适用于系统进程, 比如 surfaceflinger。

get_malloc_leak_info() 返回的 allocation_info 不是人类可以直接阅读的字符格式, 而是有着某种数据结构的二进制格式, 需要处理后再输出。

请参考 sf_debug_memory.patch

或者 base/core/jni/android_os_Debug.cpp.

通过

```
dumpsys SurfaceFlinger
```

进行触发。详细请看 3.4 内容。

另, `get_malloc_leak_info()` 返回的 `allocation_info` 中的 `info_of_allocation_entry` 的格式, 在 Android 7.1 中有变化, 具体请参考

https://android.googlesource.com/platform/bionic/+/master/libc/malloc_debug/README_api.md,

"Format of info Buffer".

2.3 html 文件分析

以本文档附带的两份 html 文件进行说明。

Native allocation HTML viewer

```
1. 0 0.00% 0 app
2. 6485632 100.00% 5095 zygote
   2054342 31.68% 2273 surfaceflinger do_arm64_start art/sigchainlib/sigchain.cc:?
   894821 13.80% 194 libc.so __libc_init /proc/self/cwd/bionic/libc/bionic/libc_init_dynamic.cpp:109
   382880 5.90% 40 libsurfaceflinger.so android::Looper::pollOnce(int) /proc/self/cwd/system/core/include/utils/Looper.h:265
   333888 5.15% 143 surfaceflinger main /proc/self/cwd/frameworks/native/services/surfaceflinger/main_surfaceflinger.cpp:56 (discriminator 1)
   323800 4.99% 73 libsurfaceflinger.so android::SurfaceFlinger::init()
   /proc/self/cwd/frameworks/native/services/surfaceflinger/SurfaceFlinger_hwcl.cpp:666
   271008 4.18% 7 libsurfaceflinger.so handler /proc/self/cwd/frameworks/native/services/surfaceflinger/SurfaceFlinger_hwcl.cpp:3986
   244032 3.76% 23 libutils.so android::Looper::pollOnce(int, int*, int*, void**) /proc/self/cwd/system/core/libutils/Looper.cpp:219
   225280 3.47% 29 libsurfaceflinger.so android::SurfaceFlingerConsumer::updateTexImage(android::SurfaceFlingerConsumer::BufferRejecter*,
   android::DisplayDevice const&, bool*, bool*, unsigned long)
```

注意 `heap_info1.html` 文件的这个函数

对比 `heap_info2.html` 文件的相同函数

Native allocation HTML viewer

```
1. 0 0.00% 0 app
2. 9370634 100.00% 5306 zygote
   2316392 24.72% 74 libsurfaceflinger.so android::Looper::pollOnce(int) /proc/self/cwd/system/core/include/utils/Looper.h:265
   2073222 22.12% 2352 surfaceflinger do_arm64_start art/sigchainlib/sigchain.cc:?
   894565 9.55% 191 libc.so __libc_init /proc/self/cwd/bionic/libc/bionic/libc_init_dynamic.cpp:109
   786432 8.39% 12 libsurfaceflinger.so android::Layer::draw(android::sp<android::DisplayDevice const> const&, bool) const
   /proc/self/cwd/frameworks/native/services/surfaceflinger/Layer.cpp:1139 (discriminator 3)
   336208 3.59% 6 libsurfaceflinger.so handler /proc/self/cwd/frameworks/native/services/surfaceflinger/SurfaceFlinger_hwcl.cpp:3986
   333888 3.56% 143 surfaceflinger main /proc/self/cwd/frameworks/native/services/surfaceflinger/main_surfaceflinger.cpp:56 (discriminator 1)
   323800 3.46% 73 libsurfaceflinger.so android::SurfaceFlinger::init()
   /proc/self/cwd/frameworks/native/services/surfaceflinger/SurfaceFlinger_hwcl.cpp:666
```

能够明显发现内存增长, 然后点击后可以查看堆栈。分析具体位置。

3 在 surfaceflinger 进程中定位内存泄漏的例子

3.1 用例和问题现象

3399_7.1, `primary_display` 使用 `hwc_composition`, HDMI 的 `external_display` 使用 `gles_composition`; 运行游戏 "激流快艇 2"; 周期执行 `procmem <pid_of_surfaceflinger>`, 发现 `sf` 进程的 `Rss` 和 `Pss` 不断增大。

3.2 调试策略

希望在任意时刻可以获取 sf 进程的全部 allocation_info, 并保存为文件.

在用例开始执行一小段时间之后, 获取一次 allocation_info, 记为 allocation_info_of_sf_1.

用例执行略长一段时间之后, 再获取一次 allocation_info, 记为 allocation_info_of_sf_2.

因为存在 leak, allocation_info_of_sf_2 中的 allocation_entry 一定比 allocation_info_of_sf_1 多, 将新增的 entries 记为 new_allocation_entries.

在 new_allocation_entries 中, 挑选出 出现次数多且 size 较大的 entry, 作为 most_suspicious_leaked_entry.

根据其中的 backtrace 信息, 找到对应的源码位置, 结合代码逻辑, 配合其他调试方式, 尝试确认问题.

3.3 实现获取 allocation_info 的 patch

为了能在任意时刻获取 allocation_info, 在 surfaceflinger 的 dump() 中加入 "获取 allocation_info, 格式转换, 输出为 文件" 的代码. 具体请参见

```
0001-DEBUG-sf-dump-allocation_info-into-a-file-in-Surface.patch.
```

集成本 patch 之后, 就可以使用 "dumsys SurfaceFlinger" 触发一次 "dump allocation_info 到文件" 的操作.

3.4 获取 dump_files_of_allocation_info 的具体步骤

集成上面的 patch, 并对应更新 libsurfaceflinger.so 到设备之后, 获取不同时间点 allocation_info_of_sf 的具体步骤如下:

1. 在 host 上执行如下命令, 使能对 sf 进程记录 allocation_info:

```
adb shell stop
```

```
adb shell setprop libc.debug.malloc.options backtrace
```

```
#注释 make malloc_debug only effective for surfaceflinger process
```



```
adb shell setprop libc.debug.malloc.program surfaceflinger
```

```
adb shell start
```

2. 执行测试用例, 等待其执行一小段时间.

3. 触发 sf dump,

```
adb shell dumysys SurfaceFlinger
```

allocation_info 将被写入到文件 /data/allocation_info.dump 中.

4. 将文件 /data/allocation_info.dump pull 到 host (PC) 上, 修改文件名保存, 记为

dump_file_of_allocation_info_of_sf_1.

5. 等待用例执行较长一段时间,

6. 再次触发 sf dump,

7. pull 并保存 当前的 dump_file_of_allocation_info_of_sf_2.

8. 参照 2.2 的说明, 使用 py 脚本转换两份 dump_file_of_allocation_info_of_sf 来进行分析

9. 转换后的 heap 文件分析可以参考 2.3 和 4.3, 3.5 比较难理解, 仅作参考。

3.5 对 dump_files_of_allocation_info 的比较分析

使用用 "Beyond Compare" 比较 dump_file_of_allocation_info_of_sf_2 和 dump_file_of_allocation_info_of_sf_1, 在 new_allocation_entries 中挑选出 moste_suspicious_leaked_entry.

在我找到的 moste_suspicious_leaked_entry 的 backtrace, 解析得到的 call_stack 中有这样的栈帧 :

```
#07      pc      000000000035e808      /system/vendor/lib64/egl/libGLES_mali.so
(eglCreateSyncKHR+700)
```

判定泄漏发生在 libGLES_mali.so 中.

实际上可以进一步定位到 libGLES_mali.so 中最终直接调用 malloc 的源码. 但是, 目前尚未确认该 leak 的原因, 正配合 support_mali 调查中.

4 Cts skqp 测试内存泄漏的例子

4.1 用例和问题现象

3399 9.0 cts 测试 skqp 时存在内存泄漏，测试过程中 skia 由于内存不足存在随机测试项 crash 的行为，32 位测试更明显，具体 log 如下：

```
D skia :external/skqp/include/core/SkBitmap.h:499: fatal error:
"assert(this->tryAllocPixels(info, rowBytes))"
```

测试项测试命令为：

```
run cts -o --skip-all-system-status-check -m CtsSkqpTestCases
```

4.2 调试步骤

通过命令行方式获取 malloc_debug 信息

```
adb shell stop
adb shell setprop libc.debug.malloc.options backtrace
adb shell start
```

在测试项开始执行一小段时间之后，获取一次 heap_info，保存文件为 heap1.txt

用测试项执行略长一段时间之后，再获取一次 heap_info，保存文件为 heap2.txt

```
adb shell am dumpheap -n <PID_TO_DUMP> /data/local/tmp/heap.txt
```

将 heap.txt 从板级拉出，通过 native_heap_viewer.py 的工具进行转换。

```
python development/scripts/native_heapdump_viewer.py --symbols
<sdk_path>/out/target/product/rk3399/symbols/ heap.txt > heap_info.txt
```

4.3对 heap_info.txt 的比较分析

76	BYTES	%TOTAL	%PARENT	COUNT	ADDR	LIBRARY	FUNCTION	LOCATION
77	115619679	100.00%	98.88%	15324	APP			
78	104695672	90.78%	90.78%	1908	7a61db851c	/data/app/org.skia.skgp-kf2kMGrtrRq7-rlh4sf2w0w--/lib/arm64/libskqp.app.so	???	???
79	104694528	90.55%	99.75%	1907	7a61db4e00	/data/app/org.skia.skgp-kf2kMGrtrRq7-rlh4sf2w0w--/lib/arm64/libskqp.app.so	???	???
80	104694528	90.55%	100.00%	1907	7a61dbf8c4	/data/app/org.skia.skgp-kf2kMGrtrRq7-rlh4sf2w0w--/lib/arm64/libskqp.app.so	???	???
81	104694528	90.55%	100.00%	1907	7a61dd1944	/data/app/org.skia.skgp-kf2kMGrtrRq7-rlh4sf2w0w--/lib/arm64/libskqp.app.so	???	???
82	104575176	90.45%	99.89%	1826	7a61dd1e4	/data/app/org.skia.skgp-kf2kMGrtrRq7-rlh4sf2w0w--/lib/arm64/libskqp.app.so	???	???
83	104575176	90.45%	100.00%	1826	7a6b71e0d8	/vendor/lib64/egl/libGLES mali.so gles2 program link program hardware/rockchip/mali_so/driver/product/gles/src/program/mali_gles2_program_api.c:608		
84	104575176	90.45%	100.00%	1826	7a6b723b04	/vendor/lib64/egl/libGLES mali.so gles2 program link program hardware/rockchip/mali_so/driver/product/gles/src/program/mali_gles2_program_interna		
85	103383824	89.42%	98.88%	1592	7a6b6bd474	/vendor/lib64/egl/libGLES mali.so cpom_shader_blob_write hardware/rockchip/mali_so/driver/product/cpom/src/mali_cpom_program_object.c:687		
86	103383824	89.42%	100.00%	1592	7a6b6c485c	/vendor/lib64/egl/libGLES mali.so cpom_serialize_stream_new hardware/rockchip/mali_so/driver/product/cpom/src/mali_cpom_serialize_tlib.c:136		
87	27073312	23.42%	26.19%	415	7a6b9d9f10	/system/lib64/vndk-sp-28/libz.so deflateInit2_external/zlib/src/deflate.c:323		
88	27073312	23.42%	100.00%	415	7a6b9dd5dc	/vendor/lib64/egl/libGLES mali.so cmem_hmem_heap_alloc hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_hmem.c:1288		
89	27073144	23.42%	100.00%	414	7a6b9e48f4	/vendor/lib64/egl/libGLES mali.so cmem_heap_alloc hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_private_heap.c:3174		
90	27072976	23.42%	100.00%	413	7a6b9e4bdc	/vendor/lib64/egl/libGLES mali.so create_chunk hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_private_heap.c:1694		
91	27072976	23.42%	100.00%	413	7a6b9df1f4	/vendor/lib64/egl/libGLES mali.so cmem_subboard_alloc hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_hoard.c:5090		
92	27072976	23.42%	100.00%	413	7a6b9df994	/vendor/lib64/egl/libGLES mali.so cmem_hoardp_hoard_hunk_create hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_hoard		
93	27072976	23.42%	100.00%	413	7a6b9e1cd0	/vendor/lib64/egl/libGLES mali.so stdlib_malloc hardware/rockchip/mali_so/driver/product/stdlib/platform_dummy/plat/mali_stdlib		
94	168	0.00%	0.00%	1	7a6b9e5590	/vendor/lib64/egl/libGLES mali.so create_block hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_private_heap.c:1486		
95	168	0.00%	100.00%	1	7a6b9d9f10	/system/lib64/vndk-sp-28/libz.so deflateInit2_external/zlib/src/deflate.c:323		
96	168	0.00%	100.00%	1	7a6b9d9f10	/system/lib64/vndk-sp-28/libz.so deflateInit2_external/zlib/src/deflate.c:323		
97	168	0.00%	100.00%	1	7a6b9d9f10	/system/lib64/vndk-sp-28/libz.so deflateInit2_external/zlib/src/deflate.c:323		
98	168	0.00%	0.00%	1	7a6b9e48f4	/vendor/lib64/egl/libGLES mali.so cmem_heap_alloc hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_private_heap.c:3174		
99	168	0.00%	100.00%	1	7a6b9e48f4	/vendor/lib64/egl/libGLES mali.so cmem_heap_alloc hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_private_heap.c:3174		
100	168	0.00%	100.00%	1	7a6b9e48f4	/vendor/lib64/egl/libGLES mali.so cmem_heap_alloc hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_private_heap.c:3174		
101	168	0.00%	100.00%	1	7a6b9e48f4	/vendor/lib64/egl/libGLES mali.so cmem_heap_alloc hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_private_heap.c:3174		
102	168	0.00%	100.00%	1	7a6b9e48f4	/vendor/lib64/egl/libGLES mali.so cmem_heap_alloc hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_private_heap.c:3174		
103	168	0.00%	100.00%	1	7a6b9e48f4	/vendor/lib64/egl/libGLES mali.so cmem_heap_alloc hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_private_heap.c:3174		
104	26614516	23.02%	100.00%	409	7a6b9d9f10	/system/lib64/vndk-sp-28/libz.so deflateInit2_external/zlib/src/deflate.c:323		
105	26614516	23.02%	100.00%	409	7a6b9dd5dc	/vendor/lib64/egl/libGLES mali.so cmem_hmem_heap_alloc hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_hmem.c:1288		
106	26614516	23.02%	100.00%	409	7a6b9e48f4	/vendor/lib64/egl/libGLES mali.so cmem_heap_alloc hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_private_heap.c:3174		
107	26614112	23.02%	100.00%	406	7a6b9e4bdc	/vendor/lib64/egl/libGLES mali.so create_chunk hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_private_heap.c:1694		
108	26614112	23.02%	100.00%	406	7a6b9df1f4	/vendor/lib64/egl/libGLES mali.so cmem_subboard_alloc hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_hoard.c:5090		
109	26614112	23.02%	100.00%	406	7a6b9df994	/vendor/lib64/egl/libGLES mali.so cmem_hoardp_hoard_hunk_create hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_hoard		
110	26614112	23.02%	100.00%	406	7a6b9e1cd0	/vendor/lib64/egl/libGLES mali.so stdlib_malloc hardware/rockchip/mali_so/driver/product/stdlib/platform_dummy/plat/mali_stdlib		
111	504	0.00%	0.00%	3	7a6b9e48f4	/vendor/lib64/egl/libGLES mali.so cmem_heap_alloc hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_private_heap.c:3174		
112	504	0.00%	100.00%	3	7a6b9e48f4	/vendor/lib64/egl/libGLES mali.so cmem_heap_alloc hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_private_heap.c:3174		
113	504	0.00%	100.00%	3	7a6b9e48f4	/vendor/lib64/egl/libGLES mali.so cmem_heap_alloc hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_private_heap.c:3174		
114	504	0.00%	100.00%	3	7a6b9e48f4	/vendor/lib64/egl/libGLES mali.so cmem_heap_alloc hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_private_heap.c:3174		
115	24976152	21.60%	100.00%	380	7a6b9d9f10	/system/lib64/vndk-sp-28/libz.so deflateInit2_external/zlib/src/deflate.c:323		
116	24976152	21.60%	100.00%	380	7a6b9dd5dc	/vendor/lib64/egl/libGLES mali.so cmem_hmem_heap_alloc hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_hmem.c:1288		
117	24975984	21.60%	100.00%	385	7a6b9e48f4	/vendor/lib64/egl/libGLES mali.so cmem_heap_alloc hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_private_heap.c:3174		
118	24975312	21.60%	100.00%	381	7a6b9e4bdc	/vendor/lib64/egl/libGLES mali.so create_chunk hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_private_heap.c:1694		
119	24975312	21.60%	100.00%	381	7a6b9df1f4	/vendor/lib64/egl/libGLES mali.so cmem_subboard_alloc hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_hoard.c:5090		
120	24975312	21.60%	100.00%	381	7a6b9df994	/vendor/lib64/egl/libGLES mali.so cmem_hoardp_hoard_hunk_create hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_hoard		
121	24975312	21.60%	100.00%	381	7a6b9e1cd0	/vendor/lib64/egl/libGLES mali.so stdlib_malloc hardware/rockchip/mali_so/driver/product/stdlib/platform_dummy/plat/mali_stdlib		

该文件的分析需要注意，第一列为 alloc 的数据量，第二列为百分比占比。实际数据和堆栈对应，方便理解可以将数据区分为三个部分，红框部分为 app 当前所申请的数据总量，蓝框部分为占用 app 申请总内存最多的数据及其对应的堆栈，分析内存泄漏需要关注申请内存最多的该部分，该 case 中蓝框部分已经达到了 90%，剩余 10% 在文件末尾，暂时不需要关心。绿框部分为显式能看出来是在做 alloc 动作的部分堆栈，该部分的总和等于蓝框部分的数据量，分析实际 alloc 动作的堆栈，都是由 cpom_serialize_stream_new 该函数为实际入口。

使用 "Beyond Compare" 比较 heap_info1.txt 和 heap_info2.txt。

76	BYTES	%TOTAL	%PARENT	COUNT	ADDR	LIBRARY	FUNCTION	LOCATION
77	115619679	100.00%	98.88%	15324	APP			
78	104695672	90.78%	90.78%	1908	7a61db851c	/data/app/org.skia.skgp-kf2kMGrtrRq7-rlh4sf2w0w--/lib/arm64/libskqp.app.so	???	???
79	104694528	90.55%	99.75%	1907	7a61db4e00	/data/app/org.skia.skgp-kf2kMGrtrRq7-rlh4sf2w0w--/lib/arm64/libskqp.app.so	???	???
80	104694528	90.55%	100.00%	1907	7a61dbf8c4	/data/app/org.skia.skgp-kf2kMGrtrRq7-rlh4sf2w0w--/lib/arm64/libskqp.app.so	???	???
81	104694528	90.55%	100.00%	1907	7a61dd1944	/data/app/org.skia.skgp-kf2kMGrtrRq7-rlh4sf2w0w--/lib/arm64/libskqp.app.so	???	???
82	104575176	90.45%	99.89%	1826	7a61dd1e4	/data/app/org.skia.skgp-kf2kMGrtrRq7-rlh4sf2w0w--/lib/arm64/libskqp.app.so	???	???
83	104575176	90.45%	100.00%	1826	7a6b71e0d8	/vendor/lib64/egl/libGLES mali.so gles2 program link program hardware/rockchip/mali_so/driver/product/gles/src/program/mali_gles2_program_api.c:608		
84	104575176	90.45%	100.00%	1826	7a6b723b04	/vendor/lib64/egl/libGLES mali.so gles2 program link program hardware/rockchip/mali_so/driver/product/gles/src/program/mali_gles2_program_interna		
85	103383824	89.42%	98.88%	1592	7a6b6bd474	/vendor/lib64/egl/libGLES mali.so cpom_shader_blob_write hardware/rockchip/mali_so/driver/product/cpom/src/mali_cpom_program_object.c:687		
86	103383824	89.42%	100.00%	1592	7a6b6c485c	/vendor/lib64/egl/libGLES mali.so cpom_serialize_stream_new hardware/rockchip/mali_so/driver/product/cpom/src/mali_cpom_serialize_tlib.c:136		
87	27073312	23.42%	26.19%	415	7a6b9d9f10	/system/lib64/vndk-sp-28/libz.so deflateInit2_external/zlib/src/deflate.c:323		
88	27073312	23.42%	100.00%	415	7a6b9dd5dc	/vendor/lib64/egl/libGLES mali.so cmem_hmem_heap_alloc hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_hmem.c:1288		
89	27073144	23.42%	100.00%	414	7a6b9e48f4	/vendor/lib64/egl/libGLES mali.so cmem_heap_alloc hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_private_heap.c:3174		
90	27072976	23.42%	100.00%	413	7a6b9e4bdc	/vendor/lib64/egl/libGLES mali.so create_chunk hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_private_heap.c:1694		
91	27072976	23.42%	100.00%	413	7a6b9df1f4	/vendor/lib64/egl/libGLES mali.so cmem_subboard_alloc hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_hoard.c:5090		
92	27072976	23.42%	100.00%	413	7a6b9df994	/vendor/lib64/egl/libGLES mali.so cmem_hoardp_hoard_hunk_create hardware/rockchip/mali_so/driver/product/cmem/src/mali_cmem_hoard		
93	27072976	23.42%	100.00%	413	7a6b9e1cd0	/vendor/lib64/egl/libGLES mali.so stdlib_malloc hardware/rockchip/mali_so/driver/product/stdlib/platform_dummy/plat/mali_stdlib		

对比数据量(BYTES)后发现主要内存泄漏实际是由 cpom_serialize_stream_new 该函数的调用引起的。