Tuning Android for low RAM

when 1 GiB is too much
License

These slides are available under a Creative Commons Attribution-ShareAlike 3.0 license. You can read the full text of the license here
http://creativecommons.org/licenses/by-sa/3.0/legalcode
You are free to

• copy, distribute, display, and perform the work

• make derivative works

• make commercial use of the work

Under the following conditions

• Attribution: you must give the original author credit

• Share Alike: if you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one (i.e. include this page exactly as it is)

• For any reuse or distribution, you must make clear to others the license terms of this work

The orginals are at http://2net.co.uk/slides/
About Chris Simmonds

- Consultant and trainer
- Working with embedded Linux since 1999
- Android since 2009
- Speaker at many conferences and workshops

"Looking after the Inner Penguin" blog at http://2net.co.uk/

https://uk.linkedin.com/in/chrisdsimmonds/

https://google.com/+chrissimmonds
Overview

- Project Svelte
- How much RAM do you need?
- Tuning Android
  - Reducing memory pressure from the Dalvik heap
  - Optimising the JIT cache
- Tuning the kernel
  - Kernel Samepage Merging (KSM)
  - Compressed swap area
The problem

- Android devices need memory for
  - Operating system
  - Background services
  - Video buffers
  - Applications

- Since Gingerbread minimum RAM has gone beyond 512 MiB

- Especially since Jelly Bean 4.1 "project Butter" which improved graphics performance by adding more buffers. But display buffers are getting larger...
Project Svelte

- Kit Kat 4.4 introduced "project Svelte": Android on devices with 512 MiB RAM

- Project Svelte consists of
  - Various memory-saving changes to Android framework
  - Tuning knobs for Android
  - Validated techniques for tuning Linux
  - Improved memory diagnostics

- See source.android.com/devices/low-ram.html

- Note: In many cases there is a tradeoff between reducing memory use and increasing CPU load (and therefore increasing power requirements)
How much RAM am I using?

- Tricky question!
- Some (most) is used by processes: apps and system daemons
  - But note that processes share a lot of read-only data
- Some is cached
  - But caches can be dropped, so cached memory is usually regarded as "free"
- Some is allocated by the kernel and not owned by any process
- Some is used for the code and data segments of the kernel and kernel modules
Memory metrics for processes

- For each Linux process we can measure
  - Vss = virtual set size: pages mapped by this process
  - Rss = resident set size: pages mapped to real memory
  - Uss = unique set size: pages of memory not shared with any other process
  - Pss = proportional set size: pages shared with other processes, divided by the number of processes sharing
- Perhaps a diagram would help...
Memory metrics

Vss = A + B + C
Rss = A + B
Uss = A
Pss = A + B/n where n is the number of processes sharing
How to calculate Pss

Pss(1) = 2 + 3/3 + 2/2 = 4
Pss(2) = 2 + 3/3 + 2/2 = 4
Pss(3) = 2 + 3/3 = 3
Sum(Pss) = 11 = total of pages in use
Tools: procrank

Part of the Android tool set for a long time: ranks processes by Pss (default), type procrank -h for more options

Example (edited):

```bash
# procrank

<table>
<thead>
<tr>
<th>PID</th>
<th>Vss</th>
<th>Rss</th>
<th>Pss</th>
<th>Uss</th>
<th>cmdline</th>
</tr>
</thead>
<tbody>
<tr>
<td>3351</td>
<td>1058776K</td>
<td>163952K</td>
<td>141197K</td>
<td>139596K</td>
<td>com.google.earth</td>
</tr>
<tr>
<td>2616</td>
<td>943156K</td>
<td>116020K</td>
<td>93360K</td>
<td>91724K</td>
<td>com.android.vending</td>
</tr>
<tr>
<td>539</td>
<td>990756K</td>
<td>112504K</td>
<td>91393K</td>
<td>89808K</td>
<td>com.android.systemui</td>
</tr>
<tr>
<td>4657</td>
<td>995760K</td>
<td>105964K</td>
<td>77829K</td>
<td>70776K</td>
<td>com.rovio.angrybirds</td>
</tr>
</tbody>
</table>

...  

<table>
<thead>
<tr>
<th>PID</th>
<th>Vss</th>
<th>Rss</th>
<th>Pss</th>
<th>Uss</th>
<th>cmdline</th>
</tr>
</thead>
<tbody>
<tr>
<td>119</td>
<td>31904K</td>
<td>7676K</td>
<td>6038K</td>
<td>5900K</td>
<td>/system/bin/surfaceflinger</td>
</tr>
<tr>
<td>122</td>
<td>27468K</td>
<td>3788K</td>
<td>3045K</td>
<td>2964K</td>
<td>/system/bin/mediaserver</td>
</tr>
</tbody>
</table>

...  

<table>
<thead>
<tr>
<th>PID</th>
<th>Vss</th>
<th>Rss</th>
<th>Pss</th>
<th>Uss</th>
<th>cmdline</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>865084K</td>
<td>24308K</td>
<td>2263K</td>
<td>860K</td>
<td>zygote</td>
</tr>
</tbody>
</table>

...  

------ ------ ------ 

717098K 669272K TOTAL

RAM: 1124832K total, 105528K free, 3808K buffers, 136624K cached, 656K shmem, 23656K slab
Another tried and tested tool: shows Vss, Rss, etc for each mapping of a single process

Example (edited):

# procmem 119

<table>
<thead>
<tr>
<th></th>
<th>Vss</th>
<th>Rss</th>
<th>Pss</th>
<th>Uss</th>
<th>ShCl</th>
<th>ShDi</th>
<th>PrCl</th>
<th>PrDi</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>4K</td>
<td>0K</td>
<td>0K</td>
<td>0K</td>
<td>0K</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td></td>
</tr>
<tr>
<td>1012K</td>
<td>4K</td>
<td>4K</td>
<td>4K</td>
<td>0K</td>
<td>OK</td>
<td>OK</td>
<td>4K</td>
<td>OK</td>
<td>[stack:944]</td>
</tr>
<tr>
<td>512K</td>
<td>512K</td>
<td>512K</td>
<td>512K</td>
<td>0K</td>
<td>OK</td>
<td>OK</td>
<td>36K</td>
<td>476K</td>
<td>/dev/mali0</td>
</tr>
<tr>
<td>512K</td>
<td>512K</td>
<td>512K</td>
<td>512K</td>
<td>0K</td>
<td>OK</td>
<td>OK</td>
<td>0K</td>
<td>512K</td>
<td>/dev/mali0</td>
</tr>
<tr>
<td>516K</td>
<td>12K</td>
<td>12K</td>
<td>12K</td>
<td>0K</td>
<td>OK</td>
<td>OK</td>
<td>12K</td>
<td>0K</td>
<td>[anon:libc_malloc]</td>
</tr>
<tr>
<td>512K</td>
<td>512K</td>
<td>512K</td>
<td>512K</td>
<td>0K</td>
<td>OK</td>
<td>OK</td>
<td>224K</td>
<td>288K</td>
<td>/dev/mali0</td>
</tr>
<tr>
<td>512K</td>
<td>512K</td>
<td>512K</td>
<td>512K</td>
<td>0K</td>
<td>OK</td>
<td>OK</td>
<td>32K</td>
<td>480K</td>
<td>/dev/mali0</td>
</tr>
<tr>
<td>516K</td>
<td>12K</td>
<td>12K</td>
<td>12K</td>
<td>0K</td>
<td>OK</td>
<td>OK</td>
<td>12K</td>
<td>0K</td>
<td>[anon:libc_malloc]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2680K</td>
<td>2668K</td>
<td>2668K</td>
<td>2668K</td>
<td>0K</td>
<td>OK</td>
<td>OK</td>
<td>2668K</td>
<td>0K</td>
<td>[heap]</td>
</tr>
<tr>
<td>132K</td>
<td>20K</td>
<td>20K</td>
<td>20K</td>
<td>0K</td>
<td>OK</td>
<td>OK</td>
<td>20K</td>
<td>0K</td>
<td>[stack]</td>
</tr>
<tr>
<td>0K</td>
<td>0K</td>
<td>0K</td>
<td>0K</td>
<td>0K</td>
<td>OK</td>
<td>OK</td>
<td>0K</td>
<td>0K</td>
<td>[vectors]</td>
</tr>
<tr>
<td>31904K</td>
<td>7676K</td>
<td>6039K</td>
<td>5900K</td>
<td>1760K</td>
<td>16K</td>
<td>4144K</td>
<td>1760K</td>
<td>TOTAL</td>
<td></td>
</tr>
</tbody>
</table>
The Android application life cycle

- Activity Manager grades applications by how many components (activities and services) are being used.
- Sets a per-process measure called \textit{oom\_adj}.
- \textit{oom\_adj} values are from -16 (important process) to 15 (unimportant process).
- As memory pressure increases, the kernel low memory killer starts killing processes starting with the highest \textit{oom\_adj}.
# Values for oom_adj

From `frameworks/base/services/java/com/android/server/am/ProcessList.java`

<table>
<thead>
<tr>
<th>State</th>
<th>oom_adj</th>
<th>Type of process</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>-16</td>
<td>daemons and system services</td>
</tr>
<tr>
<td>Persistent</td>
<td>-12</td>
<td>persistent apps, e.g. telephony</td>
</tr>
<tr>
<td>Foreground</td>
<td>0</td>
<td>contains the foreground activity</td>
</tr>
<tr>
<td>Visible</td>
<td>1</td>
<td>contains activities that are visible</td>
</tr>
<tr>
<td>Perceptible</td>
<td>2</td>
<td>e.g. background music playback</td>
</tr>
<tr>
<td>Service</td>
<td>5</td>
<td>contains an application service</td>
</tr>
<tr>
<td>Home</td>
<td>6</td>
<td>contains the home application</td>
</tr>
<tr>
<td>Previous</td>
<td>7</td>
<td>the previous foreground application</td>
</tr>
<tr>
<td>B Services</td>
<td>8</td>
<td>&quot;old and decrepit services&quot;</td>
</tr>
<tr>
<td>Cached</td>
<td>9..15</td>
<td>all activities and services destroyed</td>
</tr>
</tbody>
</table>
lowmemory killer thresholds

• Thresholds calculated according to screen size and total memory

• Example (from Nexus 10)

<table>
<thead>
<tr>
<th>oom_adj</th>
<th>Threshold (KiB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-16</td>
<td>49152</td>
</tr>
<tr>
<td>-12</td>
<td>49152</td>
</tr>
<tr>
<td>0</td>
<td>49152</td>
</tr>
<tr>
<td>1</td>
<td>61440</td>
</tr>
<tr>
<td>2</td>
<td>73728</td>
</tr>
<tr>
<td>3</td>
<td>86016</td>
</tr>
<tr>
<td>4</td>
<td>98304</td>
</tr>
<tr>
<td>5</td>
<td>98304</td>
</tr>
<tr>
<td>6</td>
<td>98304</td>
</tr>
<tr>
<td>7</td>
<td>98304</td>
</tr>
<tr>
<td>8</td>
<td>98304</td>
</tr>
<tr>
<td>9</td>
<td>98304</td>
</tr>
<tr>
<td>15</td>
<td>122880</td>
</tr>
</tbody>
</table>
Tools: meminfo

dumpsys meminfo takes the oom_adj value into account:

```
# dumpsys meminfo
Applications Memory Usage (kB):
Uptime: 5156998 Realtime: 70066043

Total PSS by process:
  141263 kB: com.google.earth (pid 3351 / activities)
  93354 kB: com.android.vending (pid 2616 / activities)
  92554 kB: com.android.systemui (pid 539)
...
Total PSS by OOM adjustment:
  19794 kB: Native
    6031 kB: surfaceflinger (pid 119)
...
  36427 kB: System
    36427 kB: system (pid 444)
...
  101001 kB: Persistent
    92554 kB: com.android.systemui (pid 539)
...
  362721 kB: Cached
    141263 kB: com.google.earth (pid 3351 / activities)
...
Total RAM: 1124832 kB
Free RAM: 633617 kB (362721 cached pss + 138452 cached + 132444 free)
Used RAM: 352407 kB (323895 used pss + 4304 buffers + 656 shmem + 23552 slab)
Lost RAM: 138808 kB
  Tuning: 192 (large 512), oom 122880 kB, restore limit 40960 kB (high-end-gfx)
```
Tools: meminfo

- In Kit Kat, `dumpsys meminfo` has been augmented to make the use of memory more clear
- Processes with `oom_adj >= 9` (`CACHE_APP_MIN_ADJ`) can be killed without the user noticing
- So *Free RAM* includes apps that can be discarded ("cached pss") and system buffers ("cached")
Tools: procstats

- *procstats* adds history to the measurement by integrating Pss over time
- Use to identify persistent memory hogs
- Typically shows up apps with long-running background services
- procstats has a nice graphical interface, and can be run from the command line
procstats

Settings -> Developer options -> Process Stats

- Bar is a summary of memory pressure: green=good, yellow=OK, red=bad
- For each app, shows
  - % of time it was running
  - a blue bar which is (average Pss * runtime)
Zoom in on second app:

- Contains a service `FlightUpdateService`
- Has been running 100% of the time
- Is taking 33 MiB
procstats command-line

- The raw data is available through system service *procstats*

- Dump the data using

```bash
# dumpsys procstats
AGGREGATED OVER LAST 24 HOURS:
  * system / 1000:
    TOTAL: 100% (22MB-31MB-35MB/19MB-29MB-33MB over 89)
    Persistent: 100% (22MB-31MB-35MB/19MB-29MB-33MB over 89)
  * com.android.systemui / u0a6:
    TOTAL: 100% (36MB-59MB-99MB/34MB-57MB-97MB over 89)
    Persistent: 100% (36MB-59MB-99MB/34MB-57MB-97MB over 89)
    Imp Fg: 0.00%
```

The memory numbers are
minPss-avgPss-maxPss / minUss-avgUss-maxUss
Tuning Android for low RAM

What are the options?

• Tune Activity manager
• Tune Dalvik
• Tune Apps
Tuning Android for low RAM

- Kit Kat has a global tuning parameter for low RAM: `ro.config.low_ram`
- If set to true:
  - Optimise allocations in Dalvik heap
  - Saves memory by reducing use of the GPU
  - New API `ActivityManager.isLowRamDevice()` returns true which apps can use as a hint that they should reduce memory usage: some Google apps are reportedly coded to make this check
- Reduces the Dalvik total PSS by 10 - 15% on devices with large bitmaps (such as Nexus 7 or 10)
Optimising Dalvik JIT

- JIT cache defaults to 1.5 MiB per app (on ARMv7a)
- If set too low will send the JIT into a thrashing mode
- For the really low-memory devices disable JIT by setting cache size to zero

```
PRODUCT_PROPERTY_OVERRIDES += dalvik.vm.jit.codecachesize=0
```
Wallpaper

- Ensure the default wallpaper setup on launcher is not live wallpaper
- Do not pre-install any live wallpapers
Tuning Linux for low RAM

What are the options?

• KSM
• Swap to compressed RAM
• Tune ION carveout
Linux memory reclaim

- **Background reclaim** is done by the `kswap` daemon
  - Started when free memory drops below a threshold: 2MB on a 2GB device and 636KB on a 512MB
  - Aims to keep some memory free by flushing dirty pages to disk (or invoking the low memory killer)
- **Direct reclaim** happens when a process tries to allocate memory and there are no free pages
  - blocks the calling thread while pages are freed
- Direct reclaim is bad because it can freeze the UI thread, leading to a poor UX
extra_free_kbytes

- Default kswapd threshold is rather low for Android devices
- `/proc/sys/vm/extra_free_kbytes` is a tuneable added by Google to Linux 3.4 to modify the kswapd threshold
- If set to 0 (default), Activity Manager will adjust it to 3x screen buffer
- Can be configured in platform config.xml 
  frameworks/base/core/res/res/values/config.xml
  - `config_extraFreeKbytesAbsolute` overrides the default chosen by Activity Manager: -1 keeps the default
  - `config_extraFreeKbytesAdjust` added (subtracted if negative) from the value chosen by Activity Manager
Kernel Samepage Merging (KSM)

- KSM is a kernel thread (ksmd) that runs in the background and compares pages in memory that have been marked `MADV_MERGEABLE` by user-space.
- If two pages are found to be the same, it merges them back to a single copy-on-write page.
- Balancing reduced memory usage vs more processing (greater power demand).
- Benefit depends on workload.
KSM controls

- Build kernel with \texttt{CONFIG_KSM=y} (Linux 2.6.32 or later)

- Controlled by these files in \texttt{/sys/kernel/mm/ksm}

<table>
<thead>
<tr>
<th>File</th>
<th>default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>run</td>
<td>0</td>
<td>Start ksmd thread if non-zero</td>
</tr>
<tr>
<td>sleep_millisecs</td>
<td>500</td>
<td>ms between scans</td>
</tr>
<tr>
<td>pages_to_scan</td>
<td>100</td>
<td>pages per scan</td>
</tr>
</tbody>
</table>

- Typically you add lines to your init.[name].rc to set up KSM
Does KSM work?

- Also in `/sys/kernel/mm/ksm`

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>full_scans</td>
<td>how many times all mergeable areas have been scanned</td>
</tr>
<tr>
<td>pages_shared</td>
<td>how many shared pages are being used</td>
</tr>
<tr>
<td>pages_sharing</td>
<td>how many more sites are sharing them i.e. how much saved</td>
</tr>
<tr>
<td>pages_unshared</td>
<td>how many pages unique but repeatedly checked for merging</td>
</tr>
<tr>
<td>pages_volatile</td>
<td>how many pages changing too fast to be placed in a tree</td>
</tr>
</tbody>
</table>

- And, at the end of `dumpsys meminfo`:

```
# dumpsys meminfo
...
KSM: 33992 kB saved from shared 4216 kB
   234796 kB unshared; 532028 kB volatile
```
Compressed swap area

- Use a compressed RAM swap area, zram, for swap
- Unused dirty pages can be swapped out and compressed
- Compression ratios in the 30-50% range are usually observed
- Once again, you are balancing reduced memory usage vs more processing (greater power demand)
Compressed swap area

• Add to kernel config

```
CONFIG_SWAP
CONFIG_CGROUP_MEM_RES_CTLR
CONFIG_CGROUP_MEM_RES_CTLR_SWAP
CONFIG_ZRAM
```

• Add to fstab:

```
/dev/block/zram0 none swap defaults
zramsize=<size in bytes>,swapprio=<swap partition priority>
```
Contiguous memory buffers

- Some simple peripherals require contiguous memory
- Typically, a region on memory is reserved using CMA
- ... and allocated using an ION carveout heap
- ... so it makes sense to review and minimise the use of such heaps
• Questions?