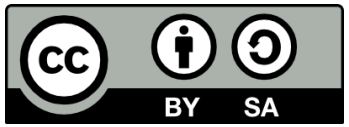
Tuning Android for low RAM

when 1 GiB is too much

2net

License



The orginals are at http://2net.co.uk/slides/android-lowmemory-abs-2014.pdf

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)

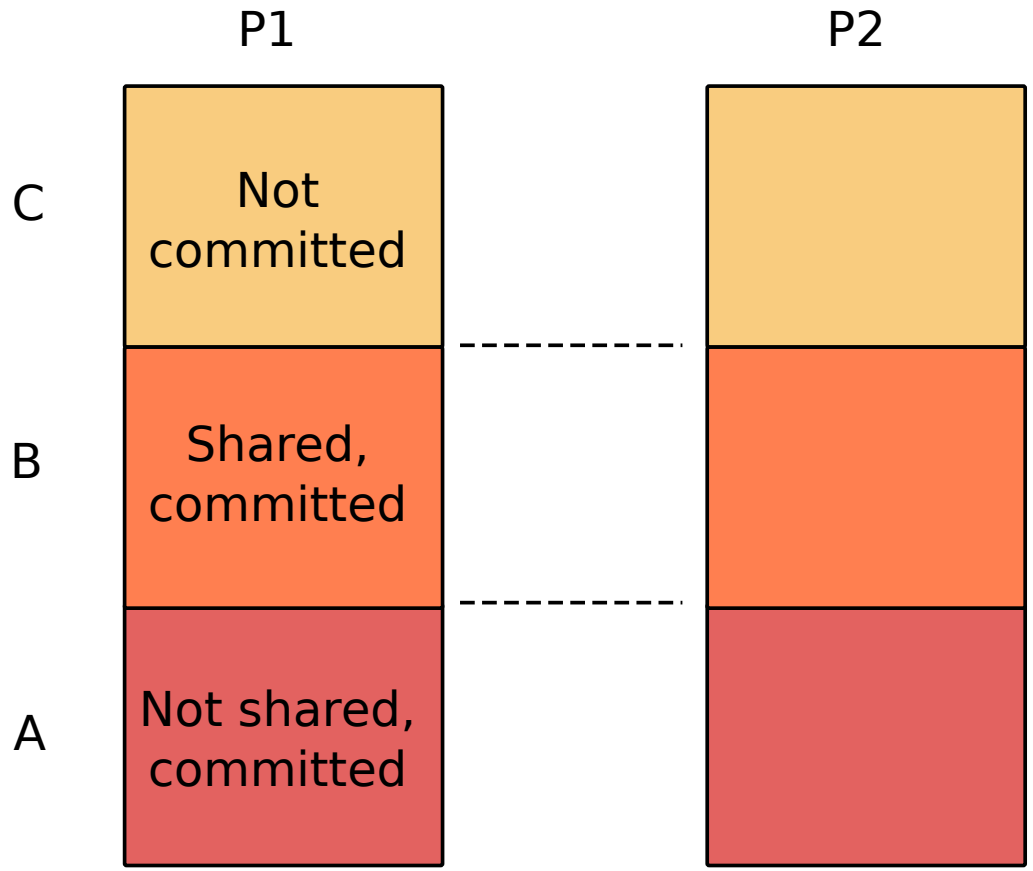
How much RAM am I using?

- 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

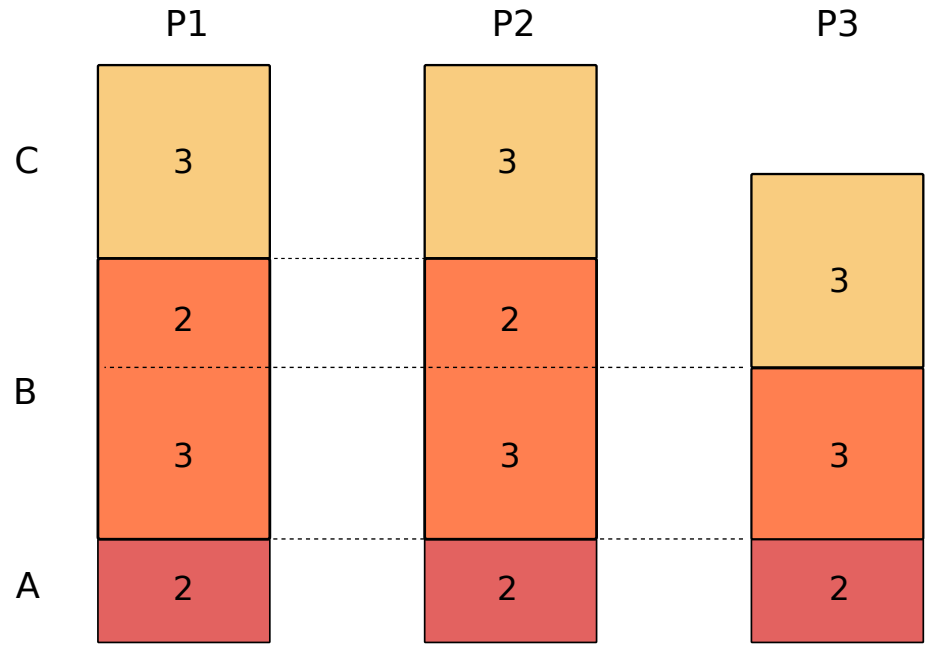
- For each Linux process we can measure
  - Vss = virtual set size: pages mapped by this process
  - Rss = resident set size: pages mapped to real memory
  - Uss = unique set size: pages of memory not shared with any other process
  - Pss = proportional set size: pages shared with other processes, divided 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):

```
# procrank
  PID       Vss      Rss      Pss      Uss  cmdline
 3351  1058776K  163952K  141197K  139596K  com.google.earth
 2616   943156K  116020K   93360K   91724K  com.android.vending
  539   990756K  112504K   91393K   89808K  com.android.systemui
 4657   995760K  105964K   77829K   70776K  com.rovio.angrybirds
...
  119    31904K    7676K    6038K    5900K  /system/bin/surfaceflinger
  122    27468K    3788K    3045K    2964K  /system/bin/mediaserver
...
  120   865084K   24308K    2263K     860K  zygote
...
                            ------   ------  ------
                           717098K  669272K  TOTAL

RAM: 1124832K total, 105528K free, 3808K buffers, 136624K cached,
 656K shmem, 23656K slab
```


Tools: procmem

Another tried and tested tool: shows Vss, Rss, etc for each mapping of a single process
Example (edited):

```
# procmem 119
    Vss      Rss      Pss      Uss     ShCl     ShDi     PrCl     PrDi  Name
-------  -------  -------  -------  -------  -------  -------  -------
     4K       0K       0K       0K       0K       0K       0K       0K
  1012K       4K       4K       4K       0K       0K       4K       0K  [stack:944]
   512K     512K     512K     512K       0K       0K      36K     476K  /dev/mali0
   512K     512K     512K     512K       0K       0K       0K     512K  /dev/mali0
   516K      12K      12K      12K       0K       0K      12K       0K  [anon:libc_malloc]
   512K     512K     512K     512K       0K       0K     224K     288K  /dev/mali0
   512K     512K     512K     512K       0K       0K      32K     480K  /dev/mali0
   516K      12K      12K      12K       0K       0K      12K       0K  [anon:libc_malloc]
...
  2680K    2668K    2668K    2668K       0K       0K    2668K       0K  [heap]
   132K      20K      20K      20K       0K       0K      20K       0K  [stack]
     0K       0K       0K       0K       0K       0K       0K       0K  [vectors]
-------  -------  -------  -------  -------  -------  -------  -------
 31904K    7676K    6039K    5900K    1760K      16K    4144K    1760K  TOTAL
```


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 *oom_adj*
- 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 oom_adj

Values for oom_adj

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

State	oom_adj	Type of process
System	-16	daemons and system services
Persistent	-12	persistent apps, e.g. telephony
Foreground	0	contains the foreground activity
Visible	1	contains activities that are visible
Perceptible	2	e.g. background music playback
Service	5	contains an application service
Home	6	contains the home application
Previous	7	the previous foreground application
B Services	8	"old and decrepit services"
Cached	9..15	all activities and services destroyed

lowmemory killer thresholds

- Thresholds calculated according to screen size and total memory
- Example (from Nexus 10)

oom_adj	Threshold (KiB)
-16	49152
-12	49152
0	49152
1	61440
2	73728
3	86016
4	98304
5	98304
6	98304
7	98304
8	98304
9	98304
15	122880

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

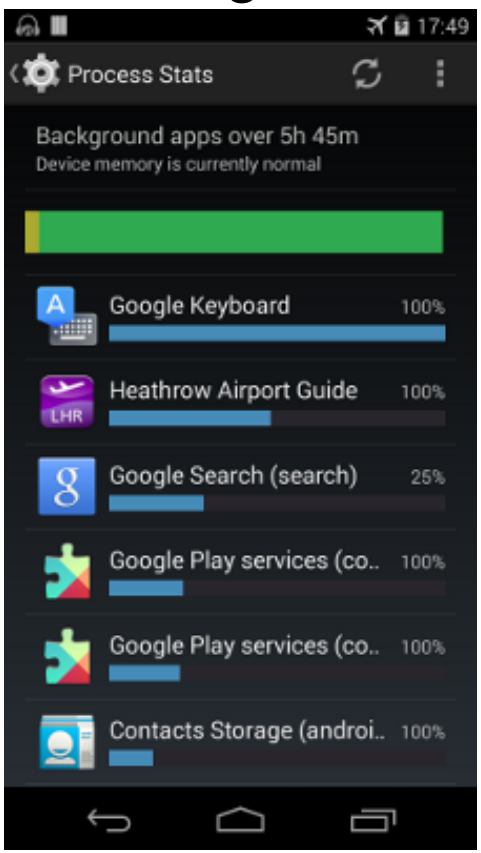
- In Kit Kat, *dumpsys meminfo* has been augmented to make the use of memory more clear
- Processes with oom_adj >= 9 (`CACHED_APP_MIN_ADJ`) can be killed without the user noticing
- So *Free RAM* includes apps that can be discarded ("cached pss") and system buffers ("cached")

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

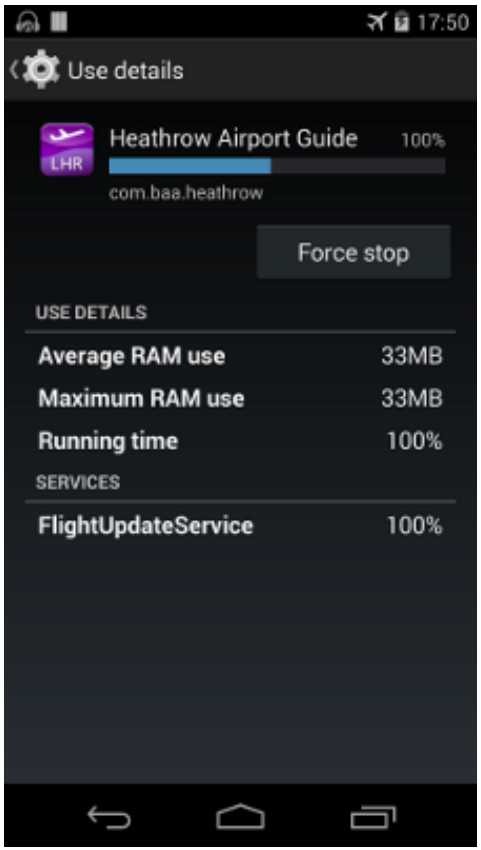
Settings -> Developer options -> Process Stats



- 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)

procstats

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 avaiable through system service *procstats*
- Dump the data using

```
 # 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)

Dalvik JIT code cache

- JIT cache defaults to 1.5 MiB per app (on ARMv7a): a typical app uses 100 KiB to 200 KiB
- You can reduce it, but if set too low will send the JIT into a thrashing mode
- For really low-memory devices it is better to disable JIT completely by setting cache size to zero

```
BoardConfig.mk:

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

- Saving: 3 MiB to 6 MiB

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 3 x 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 `CONFIG_KSM=y` (Linux 2.6.32 or later)
- Controlled by these files in `/sys/kernel/mm/ksm`

File	default	Description
run	0	Start ksmd thread if non-zero
sleep_millisecs	500	ms between scans
pages_to_scan	100	pages per scan

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

Does KSM work?

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

File	Description
full_scans	# times all mergeable pages have been scanned
pages_shared	# shared pages are being used
pages_sharing	# more sites are sharing them i.e. how much saved
pages_unshared	# pages unique but repeatedly checked for merging
pages_volatile	# pages changing too fast to be merged

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

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

- Typical saving: about 10% of used RAM

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)

Configuring zram

- Add to kernel config

```
CONFIG_SWAP
CONFIG_CGROUP_MEM_RES_CTLR
CONFIG_CGROUP_MEM_RES_CTLR_SWAP
CONFIG_ZRAM
```

- Add to /fstab.[product name]:

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

- Add to /init.[product name].rc:

```
swapon_all /fstab.[product name]
```


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?

Slides on Slide Share: http://www.slideshare.net/chrissimmonds/android-lowmemoryabs2014

