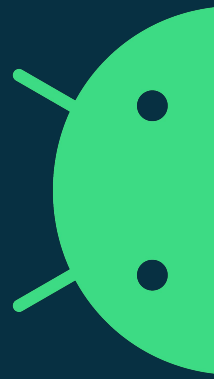
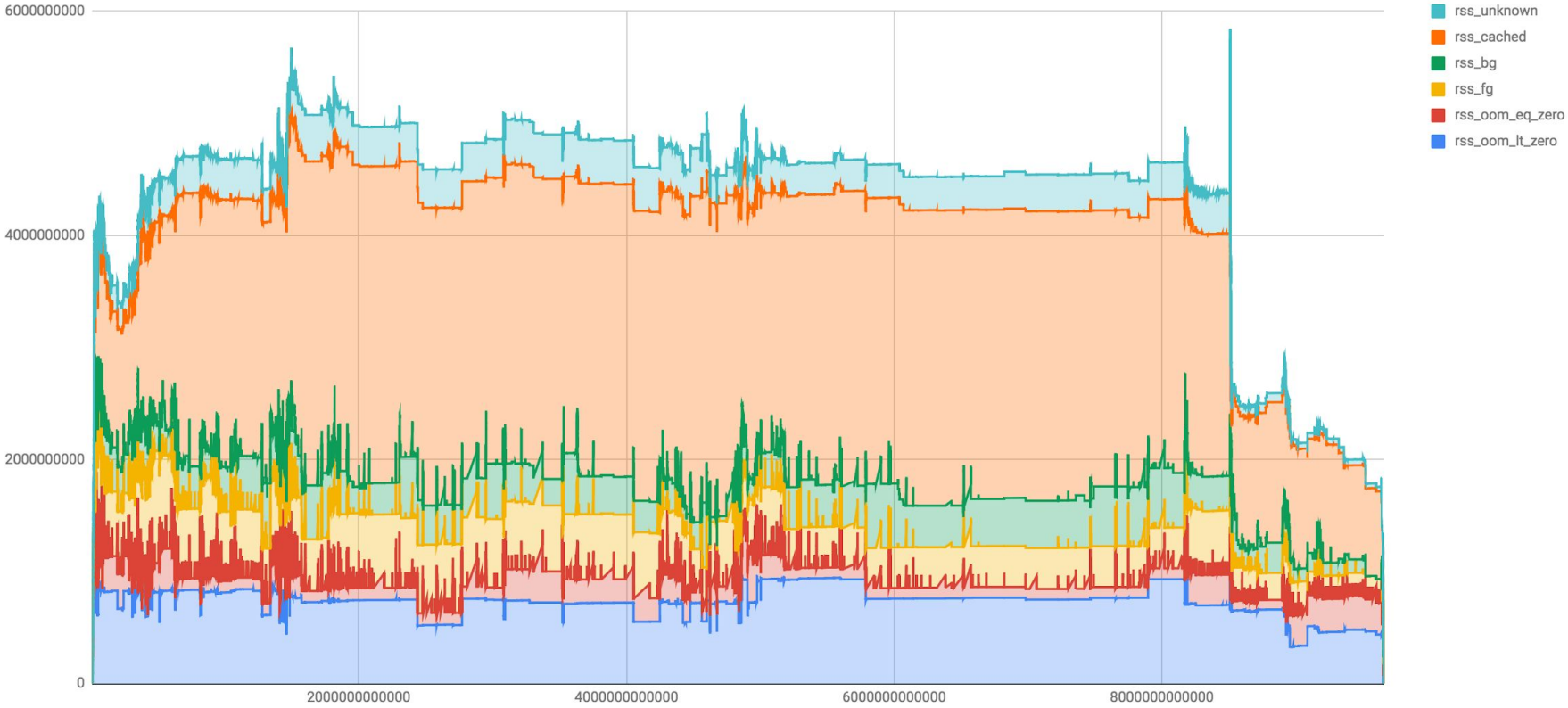
Handling memory pressure on Android

application compaction
lmkd new kill strategy
improved process tracking using pidfds



android

Where all the memory is being used?



Application compaction

Idea: keep more RAM available for interactive applications by hinting kernel about processes unlikely to be used in the near future

Solution: proactively reclaim application memory after its transition into non-interactive state

Implementation:

new process_madvise() APIs:

- MADV_COLD and MADV_PAGEOUT merged into mm tree
- process_madvice() syscall under development

Results:

- 15% less kills from the dogfood population
- up to 30% less kills while running stress tests
- no noticeable penalty on warm starts on high-end devices

process_madvise()

process_madvise() with MADV_COLD and MADV_PAGEOUT hints the kernel that process won't be used in the near future

- MADV_COLD deactivates active pages speeding up their reclaim in case of memory pressure
- MADV_PAGEOUT reclaims private pages immediately

Approach:

- Perform application compaction only when there is no memory pressure
- Deactivate file-backed pages using MADV_COLD
- Reclaim anonymous pages using MADV_PAGEOUT

New lmkd kill strategy

Approach:

- Use vmstat to detect kswapd and direct reclaim activity
- Use zone watermarks for low memory threshold calculation
- Check swap utilization to react to quick spikes in memory usage
- Check workingset_refaults to react to thrashing caused by slower increases in memory usage

Advantage:

- Unified strategy for high-performance and low-memory Android Go devices
- Decrease in the number of tunables (retires 8 old knobs while adding 4 new ones - two PSI thresholds, thrashing limit and thrashing limit decay)

Results

On high-end (Pixel 3) devices:

- 25% less kills with 15% app launch time improvement running high memory pressure tests

On low-memory (Android Go) devices:

- 23-34% app launch time improvement on tests involving small and medium size workingsets, 5-6% regression on large size workingsets
- Additional heuristic to limit active workingset size by preemptive killing might be useful

New tunables

`ro.lmk.psi_partial_stall_ms` **- psi partial stall threshold (memory pressure)**

`ro.lmk.psi_complete_stall_ms` **- psi complete stall threshold (severe memory pressure)**

`ro.lmk.swap_free_low_percentage` **- low swap threshold to react to memory spikes**

`ro.lmk.thrashing_limit` **- workingset refault threshold as % of file-backed pagecache size**

`ro.lmk.thrashing_limit_decay` **- thrashing threshold decay in % when system keeps thrashing**

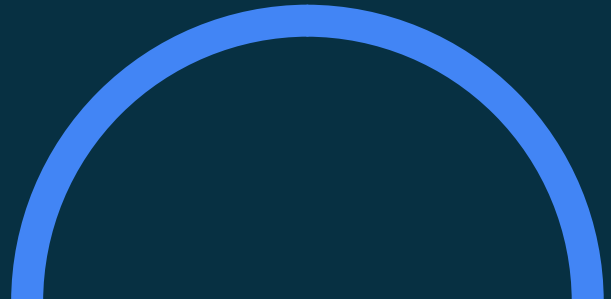
Improved process tracking using pidfds

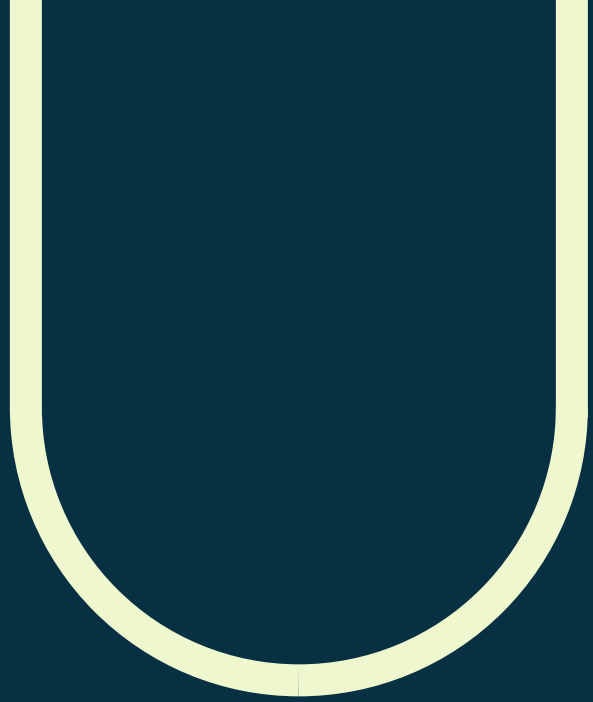
Issue: occasional pid reuse after long runtime session causing an important process being confused with an unimportant one and being killed as a result

- Occurs rarely but hard to track and troubleshoot
- Severely impacts user experience (interactive process crashes)
- No efficient way to prevent

Solution: use pidfd when process is registered and use it to check for process existence, to kill using pidfd_send_signal(), wait for its death using poll(). pidfd is not reused until all processes close the file descriptor.

Questions ?





android