Efficient Memory Management on Mobile Devices

Bartlomiej Zolnierkiewicz
Samsung R&D Institute Poland
b.zolnierkie@samsung.com

September 17, 2013

Issues on mobile systems:

- limited resources
- no physical swap
- need for custom Out-Of-Memory (OOM) handling
- custom user-space requirements

Solutions:

- control groups memory controller
- memory pressure cgroup notifier
- per-process/cgroup reclaim
- memory compression
- custom OOM handling
- memory volatile ranges
- Kernel Samepage Merging (KSM)

control groups memory controller

- control groups feature provides a mechanism for aggregating/partitioning sets of tasks, and all their future children, into hierarchical groups with specialized behaviour
- terminology:
  - control group (cgroup)
    - associates a set of tasks with a set of parameters for one or more subsystems
  - subsystem (usually a resource controller)
    - module that makes use of the task grouping facilities provided by cgroups to treat a group of tasks in particular ways
  - hierarchy
    - set of cgroups arranged in a tree, such that every task in the system is in exactly one of the cgroups in the hierarchy, and a set of subsystems
    - each hierarchy has an instance of the cgroup virtual filesystem associated with it
- cgroups memory controller (memcg) isolates the memory behavior of a group of tasks from the rest of the system
- memcg was first introduced in kernel 2.6.25

control groups memory controller (continued)

- accounts and limits usage of anonymous pages and file caches
- optional accounting and limiting of swap and kernel memory usage
- currently kernel memory usage accounting includes:
  - stack pages
  - slab pages
  - sockets memory pressure
  - tcp memory pressure
- hierarchical accounting and reclaim
  - the hierarchy is created by creating the appropriate cgroups in the cgroup filesystem
  - „root" cgroup has no limit controls
- move charges at task migration
  - uncharge task's pages from the old cgroup, charge them to the new cgroup
  - disabled by default

control groups memory controller (continued)

- soft limits
  - when the system detects memory contention or low memory, reclaim tries to push cgroups back to their soft limits
  - reclaim gets invoked from kswapd
  - best-effort feature
- usage threshold notifier
  - possible to register multiple memory thresholds and get notifications when they are crossed
  - uses eventfd mechanism
  - application gets notified when memory usage crosses threshold in any direction
  - applicable also to „root” cgroup
- memory pressure notifier
- possibility to disable in-kernel OOM-killer and get OOM notifications in user-space

memory pressure cgroup notifier

- applications can receive notifications when their specific cgroup is running low on memory, even if the system as a whole is not under memory pressure
- the feature is using eventfd mechanism to provide notifications
- three levels of memory pressure:
  - "low”
    - memory reclaim is happening at a low level
    - application doesn’t need to do anything
  - "medium”
    - some swapping is happening
    - application should clean up some low-value caches
  - "oom”
    - memory pressure is severe
    - application should clean up everything possible
- introduced in kernel 3.10
- "shrinker" interface to control user-space reclaim behavior not merged

memory pressure cgroup - "shrinker" interface

- operates on the concept of chunks of an application-defined size (i.e. 1MB)
- application passes chunk size to kernel and waits for notifications (using eventfd mechanism)
- occasionally, the application should write a string to the shrinker file indicating how many chunks have been allocated or (using a negative count) freed
- kernel uses this information to maintain an internal count of how many reclaimable chunks the application has
- when kernel wants the application to free some memory, the notification to application will come in the form of an integer count of the number of chunks that should be freed
- if the application isn't able to reclaim all chunks for some reason, it should re-add the number of chunks that were not freed

per-process reclaim

- user-space can reclaim any target process anytime
- it can avoid process killing for getting free memory
- adds new "/proc/<pid>/reclaim" knob:
  - echo file > /proc/<pid>/reclaim # reclaim file-backed pages only
  - echo anon > /proc/<pid>/reclaim # reclaim anonymous pages only
  - echo all > /proc/<pid>/reclaim # reclaim all pages
- per address space reclaim (i.e. reclaim for multi tabs in WebKit)
  - echo [addr] [size-byte] > /proc/pid/reclaim
- proposed in March 2013, currently at v5

per-cgroup reclaim

- "memory.force_reclaim" attribute
- it can reclaim the pages from the given cgroup by writing the number of pages to „memory.force_reclaim” file
- tries to reclaim the given number of pages (can reclaim more or fail)
- similar functionality can be achieved by other means:
  - set the soft limit to 0 for the target cgroup which would enforce reclaim during the next global reclaim
  - move tasks to a cgroup with hard limit reduced by the amount of pages which you want to free
- proposed in June 2013, currently NACKed

memory compression

- swap/anonymous pages compression
  - using frontswap kernel hooks (zswap, zcache1, zcache2)
    - requires at least one physical swap device
    - "caching layer" for swap pages
    - poorly-compressible pages can be send to "real" swap
    - reduces disk I/O at the cost of memory usage and CPU time
  - using block device layer (zram)
    - requires user-space configuration (via mkswap and swapon)
    - acts like a normal swap disk
    - reduces memory usage at the cost of CPU time
- clean page cache pages compression
  - using cleancache kernel hooks (zcache)
    - "caching layer" for clean page cache pages
    - reduces disk I/O at the cost of memory usage and CPU time

memory compression (continued)

- memory allocators for compressed pages
  - zbud (used by zswap and zcache2/3)
    - no more than two compressed pages are contained in a page frame
    - fragmentation is limited and pages eviction is easy
    - merged in kernel 3.11 (together with zswap)
  - zsmalloc (used by zram and zcache1)
    - replaced xvmalloc
    - high density (empirical average is 2.6 compressed pages per page frame)
    - suffers from fragmentation and pages eviction is difficult
    - in drivers/staging since kernel 3.3
- real-life tests of swap compression using Tizen applications
  - done on ARM EXYNOS platform using per-process reclaim for anon pages
  - compression ratio is about 50% (similar results with zswap/zcache2/zram)
  - it takes 3x the original reclaim time with the compression enabled
  - LZ4 compression algorithm (merged in kernel 3.11) is ~10% faster than LZO (while providing minimally worse compression ratio)

memory compression (continued)

- zswap merged in kernel 3.11
- zcache1 merged to drivers/staging in kernel 2.6.38
- zcache2 merged to drivers/staging in kernel 3.6
- zcache2 replaced zcache1 in kernel 3.8
- zcache2 to be deleted from drivers/staging in kernel 3.12
- zcache3 (cleancache support only) proposed in August 2013 for mainline
- zram (first named as ramzswap) in drivers/staging since kernel 2.6.33

custom OOM handling - ulmkd

- ulmkd = userspace low memory killer daemon
- the policy now lives in the userland
- expects some "low memory notifications" services from the kernel
- possible (though not implemented currently) to send user-specific events to processes which would i.e.:
  - release/garbage collect memory
  - suspend and put selected processes into swap
- drop-in replacement for Android lowmemorykiller driver
- consists of two parts:
  - low memory notifications handling
    - cgroups
    - vmevent (ancestor to mempressure cgroup notifications)
  - task list management
    - /proc based (daemon reads PIDs and oom_adj values)
    - shared memory based (the activity manager keeps the task list in memory)
- announced in August 2012, git://git.infradead.org/users/cbou/ulmkd.git

custom OOM handling - Tizen lowmem notifier

- mixed kernel/user-space solution -> user-space only solution
- processes are divided into foreground and background ones (UI related)
- when it comes to killing processes background ones are killed first
- depends on cgroup memory pressure notifier for low memory notifications
- currently uses (too much) hardcoded thresholds for various operations
- uses cgroup.procs information for selecting victim processes
- can put selected processes into compressed swap (zswap)
- work-in-progress

volatile ranges

- inspired by the ashmem device implementation for Android
- provide a way for user-space applications to give hints to the kernel, about memory that is not immediately in use an can be regenerated if needed
- real-life example: web browser caches
- application informs the kernel that a range of pages in its address space can be discarded at any time if the kernel needs to reclaim memory (by marking pages as volatile)
- if the application needs those pages later it can request the kernel to no longer allow purging that memory (by marking pages as non-volatile)
- if the kernel has already purged the memory when application requests it to be made non-volatile, the kernel will return a warning value to notify application that the data was lost and must be regenerated
- if application accesses memory marked volatile that has not been purged, it will get the values it expects
- if application accesses volatile memory that has been purged, the kernel will send it a SIGBUS signal (handler can mark memory as non-volatile and regenerate its content)

volatile ranges (continued)

- a new vrange() syscall that can be used for both file and anonymous pages
- int vrange(unsigned long address, size_t length, int mode, int *purged)
  - address: starting address, page aligned
  - length: length of the range to be marked, in page size units
  - mode
    - VRANGE_VOLATILE: marks the specified range as volatile and able to be purged
    - VRANGE_NONVOLATILE: marks the specified range as non-volatile, if any data in that range was volatile and has been purged, 1 will be returned in the purged pointer
  - purged
    - pointer to an integer that will be set to 1 if any data in the range being marked non-volatile has been purged and is lost; if it is zero, then no data in the specified range has been lost
- returns the number of bytes marked or unmarked (it may return fewer bytes then specified if it ran into a problem)
- on error a negative value is returned (no changes were made)
- first proposed in November 2011 (as extension to posix_fadvise() interface), currently at v8

Kernel Samepage Merging (KSM)

- allows dynamically sharing identical memory pages between one or more processes
- memory is periodically scanned by ksmd daemon, identical pages are identified and merged
- main users are systems running virtualized guests using i.e. KVM
- those guests doesn't share process-tree relationship but contain a lot of duplicated memory content
- only merges anonymous (private) pages, not page cache (file) pages
- uses two red-black trees to track memory pages
- earlier version used hash tables but there were security and patent issues
- pages that change over one memory scanning cycle are not shared
- shared pages are marked as read-only (Copy-on-Write is done on page content modification)
- introduced in kernel 2.6.32

Kernel Samepage Merging (KSM) (continued)

- KSM scanning may use a lot of processing power so it is important to limit it to areas that are likely to benefit from sharing
- application must mark the area of address space as a likely candidate for merging to enable KSM to operate on it
  - using madvise(addr, length, MADV_MERGEABLE) system call
- application may unmark the area and restore unshared pages
  - using madvise(addr, length, MADV_UNMERGEABLE) system call
  - unmerging call may require more memory than is available failing with EAGAIN or waking up OOM killer
- if KSM is not configured into the running kernel madvise() calls will fail with EINVAL
- if KSM is configured into the running kernel madvise() will normally succeed even if:
  - ksmd is not currently running
  - the range contains no pages that can be actually merged
  - unmark advice is applied to a range which was never marked as mergeable

References

- various authors, Linux Kernel documentation
  - linux/Documentation/cgroups/cgroups.txt
  - linux/Documentation/cgroups/memory.txt
  - linux/Documentation/vm/ksm.txt
- B. Zolnierkiewicz, The mempressure control group proposal
  - http://lwn.net/Articles/531077/
- Anton Vorontsov, Userspace low memory killer daemon announcement
  - https://lkml.org/lkml/2012/8/16/49
- John Stultz, Volatile Ranges v8 announcement
  - https://lkml.org/lkml/2013/6/12/6

Q&A

Thank you.