Volatile Range

2014 Embedded Linux Conference San Jose

Minchan Kim
LG Electronics
minchan at kernel dot org

Contents

- What’s the problems?
- What is volatile range
- Use cases
- Issues
- Experiment

Userspace is hungry..

- Userspace doing elastic memory management is more and more common
  - Memory ballooning
  - Low memory notifier
  - Per process reclaim
  - madvise
- Not easy to be harmonized with system without much pain with existing syscall
- Userspace has key information on what's the precious resource while kernel is dumb

Soft Takeoff

- mmap(2)
  - Create just new vm_area_struct which manage address range
  - mmap_sem with write-side lock
- Page fault
  - mmap_sem with read-side lock → page allocation → page zeroing for security→ populate page table entry
- munmap(2)
  - Kernel destroys just vm_area_struct

Munmap internal

- mmap_sem with write-side lock
- detach all of pages from page table for the range
- Page free
- O(N)
- TLB flush

Dilemma of Allocator

- General allocators are very sensitive to mmap_sem for performance POV
- So, they allocate/free huge chunk by batching to avoid frequent mmap/munmap with small chunks
- Allocators prefer madv_dontneed to munmap
- Once allocator reallocates some freed range by munmap/dontneed, user will see lots of page fault
- Keeping garbage(ie, freed by user) causes unnecessary IO for swapping and even OOM on swapless system

Munmap vs. madv_dontneed

	munmap	madv_dontneed
mmap_sem	write	read
Syscall overhead	O(N)	O(N)
TLB flush	y	y
Access after syscall	SIGSEGV	Zero

Dilemma of Cache

- Normally, caching is good thing for performance
- Every program might want it
- What happens if they consume too much memory?
- What happens if they keep too small memory?
- So, we need to balance cache size with system free memory instead of each process just considering own cache size

Questions

- When should we shrink cache?
- How much of memory should we shrink?
- Is it necessary to preserve content for reclaiming?

“Volatile range” is motivated by those

What is “Volatile Range”

- A method for userland to inform the kernel that a range of memory is safe to discard
- Hint that regeneration cost is cheaper than swap in/out cost
- Actual freeing of the memory is done under memory pressure
- User can try to cancel the action and be able to quickly access any freed pages

Man 2 vrange - old

```
ssize_t vrange(unsigned long start, size_t length, int
mode, int *purged)
```

- Mode : volatility
  - VRANGE_VOLATILE(ie, marking)
  - VRANGE_NOVOLATILE(ie, unmarking)
- int *purged : indicate whether range was purged
- If you access on purged page without unmarking volatility, you will see SIGBUS
- On success, returns the number of bytes marked or unmarked

madv_dontneed vs. vrange

	munmap	madv_dontneed	vrange
mmap_sem	write	read	read
Syscall overhead	O(N)	O(N)	O(1)
TLB flush	Y	Y	N
Access after syscall	SIGSEGV	Zero	Zero/Original or SIGBUS

Access of page after marking syscall

- If there was purge by memory pressure and you didn’t call unmarking of vrange, you will see SIGBUS
- If there was purge by memory pressure and you did call unmarking of vrange, you will see zeroed page
- If there was no purge and you didn’t call unmarking, you will see original content but you could see sudden SIGBUS in future
- If there was no purge and you did call unmarking of vrange, you see original content and you never lose the content

How to use volatile ranges

- General Allocator
  - Mark freed(3) range volatile
  - Unmark freed(3) range right before reallocation
- Browser
  - Mark invisible range of the screen volatile
  - Unmark invisible range right before scroll up/down
  - If there was purged page, browser should regenerate the contents

Decompress library

- Want to decompress a compressed library file into memory
- Mark the uncompressed code pages as volatile
- Want purge cold pages, leaving hot pages in memory
- When they traverse a purged page, they handle the SIGBUS and patch the page up

Thread Sanitizer

- Shadow memory is 4-8x in size relative to application memory
- Want to mark the shadow memory region(e.g. 70TB) as volatile at start up
- Preserve volatility after memory accesses to the range so marking is zero-frequency once start up
- Doesn't matter sudden zeroing of the page but SIGBUS is undesirable

Development History

- John Stultz in Linaro started to upstream volatile range as a feature of ashmem on Android – Nov 2011
- I thought we could use the idea to anonymous pages – Oct 2012
- 2013 LSF/MM – Apr 2013
- We started to collaborate to unify both features – Jun 2013
- It made syscall very complicated – Jan 2014

What makes vrange so complicated?

- Implementation
  - Duplicate management of the mmaped range(ie, vrange and vm_area_struct)
  - Hinting syscall should be minimal cost
  - mmap_sem scalability
  - Aging of page on swapless system
- User Requirements
  - Object aging – page vs object
  - Reclaim preference
  - SIGBUS vs Zero page

2014 LSF/MM

- Implementation
  - Use vm_area_struct only
  - Solve mmap_sem problem with madvise(MADV_FREE)
  - Aging on swapless system

- User Requirements
  - Focused on volatility(ie, reclaim preference/aging unit should be another syscall ex, MADV_COLD)
  - We really needed SIGBUS semantic?

MADV_VOLATILE

```
int madvise(void *addr, size_t length, int advise)
```

- advise : volatility
  - MADV_VOLATILE(ie, marking)
  - MADV_NOVOLATILE(ie, unmarking)
- Return value : indicate whether range was purged
- SIGBUS if access on purged page without unmarking volatility

Old vrange vs. New MADV_VOLATILE

	munmap	madv_dontneed	vrange	madv_volatile
mmap_sem	write	read	read	write
Marking syscall overhead	O(N)	O(N)	O(1)	O(1)
Unmarking syscall overhead			O(1)	O(N)
TLB flush	Y	Y	N	Y(unmarking)
Access after syscall	SIGSEGV	Zero		

Lazy free via MADV_FREE

- Delayed madv_dontneed and could be canceled by “store” operation from userside
- Unmarking syscall is “store” operation from user process
- It means you never lose recent overwrite
- Unlike MADV_VOLATILE, there is no way to detect purge and just return zero page instead of SIGBUS if purge happens
- Benefit is to avoid page fault overhead if there is no memory pressure and avoid swapping although memory pressure happens
- Remove mmap_sem problem from volatile range requirement

How to use - madv_free

- General allocators could use madv_free instead of madv_dontneed
- Garbage by free(3)ed could be discarded by kernel when memory is tight
- If allocator reallocates marked range by madv_free to user by malloc(3) and user writes a new data, user doesn’t lose the recent write
- It’s okay that other data in allocated range except recent write is zero or garbage because “man 3 malloc” doesn’t say it makes sure newly allocated space should be zero

madv_dontneed vs. madv_free

	munmap	madv_dontneed	madv_free
mmap_sem	write	read	read
Syscall overhead	O(N)	O(N)	O(N)
TLB flush	y	y	y
Access after syscall	SIGSEGV	Zero	Zero - purged Original – no purged
Reallocate overhead	Mmap(2) + page fault	Page fault	Page fault – purged None – no purged

Experiment

- Ebizzy – webserver DB workload
- 4 CPU, 2.7Hz, 2G ram, jemalloc allocator

thread	vanilla	old vrange	madvfree
1	7436	30231	15292
4	16875	56341	36320
8	16966	49239	35915

TODO

- Page aging on swapless system
- Make MADV_NOVOLATILE O(1)
- SIGBUS vs Zeroed-page
- Vrange-file support for shared memory

Acknowledgment

- John Stultz – Linaro
- Jason Evans – Facebook
- KOSAKI Motohiro – Fujitsu
- Johannes Weiner - Redhat

Questions