The slab allocators of past, present, and future

Vlastimil Babka
Linux Kernel Developer, SUSE Labs
vbabka@suse.cz
LPC 2022, Linux Kernel Memory Management MC
12 September 2022 (r1)

Slab allocators of the past

- `lib/malloc.c` – Linux-0.11 (Dec 1991), by Ted Ts'o
  - “(this one is horrible: the free needs the size for good performance, which will result in years of "`free_s()`" pains)
- `mm/kmalloc.c` – Linux-0.99.14 (Nov 1993), by Roger Wolff
  - size is in prepended `block_header`
- `mm/slab.c` – Linux-2.1.23 (Jan 1997), by Mark Hemment
  - Implementation per “UNIX Internals: The New Frontiers “ by Uresh Vahalia and “The Slab Allocator: An Object-Caching Kernel Memory Allocator” by Jeff Bonwick (Sun Microsystems)
  - `kmem_cache_alloc()/_free()`, `kmem_alloc()/_free()` with a set of “kmalloc” caches (not actually used in this version)
  - first caches: for `struct vm_area_struct`, `struct sock`
- Linux-2.1.38pre1 (Mar 1997) – `mm/kmalloc.c` deleted, `mm/slab.c` renames its `kmem_alloc()/_free()` to `kmalloc()/kfree()`

Slab allocators of the past

- Further `mm/slab.c` evolution:
  - 2.4.0-test3pre6 (June 2000) - Manfred Spraul – “major cleanup”, per-cpu arrays
  - v2.4.9.6 (Feb 2002) – Andrea Arcangelli – separate full/partial/free lists
  - 2.6.12 (June 2005) – another major cleanup – mandatory per-cpu arrays
  - 2.6.14 (Oct 2005) – Christoph Lameter – NUMA awareness
- `mm/slob.c` – v2.6.16 (March 2006), by Matt Mackall
  - “similar to the original Linux kmalloc allocator that SLAB replaced.”
  - “smaller code, more memory efficient”, but “scales poorly and suffers from fragmentation”
- `mm/slub.c` – v2.6.22 (July 2007), by Christoph Lameter
  - “motivated by the complexity of the existing code in `mm/slab.c`”
  - no array caches (“queues”), instead per-cpu slabs, less meta-data overhead, runtime tunables, cache merging, debugging, on-demand DMA caches, larger page sizes

Slab allocators of the past

- SLUB made default – v2.6.23 (Oct 2007) – off-list patch?
  - “There are some reports that 2.6.22 has SLUB as the default. Not true! This will make SLUB the default for 2.6.23.”
- Attempts to merge the best of SLUB and SLAB
  - SLQB in 2008-2009 (Nick Piggin)
    - SLUB-like freelist, struct page, SLAB-like per-cpu obj cache
    - Not merged presumably because Linus didn’t want YASA
  - SLEB – “The Enhanced(hopefully) Slab Allocator” in 2010 (C. Lameter)
    - SLUB + per-cpu object caches + bitmaps for free objects
    - Also called S+Q, Unified slab allocator in further versions
    - Some performance regressions were reported, further development discontinued (?)
- Some advantages of SLUB ported to SLAB over time
  - Use of `struct page` fields instead of extra allocated `struct slab` (freelist still separate)
  - Cache merging

Slab allocators of the present

- SLAB
  - Relatively stable implementation and performance (better for some workloads?), smaller memory usage than SLUB (?)
  - Not so useful for debugging (`CONFIG_` enabled)
- SLOB
  - Smallest memory footprint – a tradeoff for worse performance
  - Cannot `kfree()` allocations from `kmem_cache_alloc()`
- SLUB
  - Overall best performance, the best debugging features (always compiled-in, boot-time enabled)
  - Primary target for new features (`PREEMPT_RT`)

Disadvantages of multiple implementations

- More code to maintain, with varying degrees of testing
- A common layer to maintain (`mm/slab_common.c`), 1294 lines
  - More unification scheduled for 6.1 – 1435 lines
  - Code duplication vs optimized code (avoiding nested calls)
- Features compatible with only subset of implementations
  - `SLUB || SLAB` – `MEMCG_KMEM`, `FAILSLAB`, `KASAN`, `KFENCE`
    - Extra work was needed to hook e.g. `MEMCG_KMEM` and `KASAN` to both of them
  - SLUB – `KASAN_HW_TAGS`, `PREEMPT_RT`
- Blocks API improvements
  - Allowing `kfree()` for `kmem_cache_alloc()` objects would be nice, was even requested
  - SLOB would have to prepend the size header for every object (now just `kmalloc()`)
  - Larger memory footprint goes against its main advantage, made even worse due to
    - `ARCH_KMALLOC_MINALIGN` which is is up to 128 bytes on arm64
    - DMA safety guarantees of `kmalloc()` - the 128B header can't be shared with another `kmalloc()` obj

Slab allocators of the future

- Can we deprecate and drop SLAB? Some past attempts:
  - David Rientjes (2012, in a thread about defconfigs with SLAB) - “SLUB is a non-starter for us and incurs a >10% performance degradation in netperf TCP_RR.”
  - Tobin Harding (2019) – proposed removal of SLAB (reasoning: 2 year old bug causing kernel crash found, so it’s unused? On closer look, just `CONFIG_DEBUG_SLAB_LEAK`)
    - Didn’t happen as there were still users (SUSE, Google), networking workloads again mentioned
  - Hyeonggon Yoo (2021) - A question if SLAB is deprecated
    - DavidR: “Google actually uses it for its production kernel although we're investigating the performance results that we can obtain from SLUB now that we have per-object memcg accounting. There have been workloads, as you mention, that perform better with SLAB even though SLUB can make up for some of its degradation by throwing more memory at the problem (like per-cpu partial slabs).”

Slab allocators of the future

- Can we deprecate and drop SLOB?
  - It's small amount of code and not a big burden, but blocks the `kfree()` API guarantee
  - A 2021 thread by Hyeonggon Yoo - can SLUB be tuned to have comparable footprint?
    - Omit all percpu caches, use low order page allocations (didn't have much effect)…
    - My quick virtme test: `/proc/meminfo` Slab: 30800kB with SLOB, 32200kB with patched SLUB
    - Hyeonggon's test with a tiny config: 368 kB with SLOB, 552 kB with SLUB
      - With SLUB tuning: 388 kB with SLOB, 436 kB with SLUB, 408 kB when ignoring `SLAB_NEVER_MERGE`
    - The thread died away, but maybe it's promising after all?
    - Rewrite SLOB in a way that has similar footprint while knowing all object sizes and providing DMA alignment guarantees?
  - Is there any interest in "tiny Linux" (~16MB RAM) these days anymore?
    - E.g. OpenWRT devices are larger, uses SLUB anyway (?)

Thank you.