events-kmem.txt
Subsystem Trace Points: kmem

The kmem tracing system captures events related to object and page allocation within the kernel. Broadly speaking there are five major subheadings.

- o Slab allocation of small objects of unknown type (kmalloc)
- o Slab allocation of small objects of known type
- o Page allocation
- o Per-CPU Allocator Activity
- o External Fragmentation

This document describes what each of the tracepoints is and why they might be useful.

1. Slab allocation of small objects of unknown type

```
kmalloc         call_site=%lx ptr=%p bytes_req=%zu bytes_alloc=%zu gfp_flags=%s
kmalloc_node    call_site=%lx ptr=%p bytes_req=%zu bytes_alloc=%zu gfp_flags=%s
node=%d
kfree           call_site=%lx ptr=%p
```

Heavy activity for these events may indicate that a specific cache is justified, particularly if kmalloc slab pages are getting significantly internal fragmented as a result of the allocation pattern. By correlating kmalloc with kfree, it may be possible to identify memory leaks and where the allocation sites were.

2. Slab allocation of small objects of known type

```
kmem_cache_alloc        call_site=%lx ptr=%p bytes_req=%zu bytes_alloc=%zu
gfp_flags=%s
kmem_cache_alloc_node   call_site=%lx ptr=%p bytes_req=%zu bytes_alloc=%zu
gfp_flags=%s node=%d
kmem_cache_free         call_site=%lx ptr=%p
```

These events are similar in usage to the kmalloc-related events except that it is likely easier to pin the event down to a specific cache. At the time of writing, no information is available on what slab is being allocated from, but the call_site can usually be used to extrapolate that information.

3. Page allocation

```
mm_page_alloc             page=%p pfn=%lu order=%d migratetype=%d gfp_flags=%s
mm_page_alloc_zone_locked page=%p pfn=%lu order=%u migratetype=%d cpu=%d
percpu_refill=%d
mm_page_free_direct       page=%p pfn=%lu order=%d
mm_pagevec_free           page=%p pfn=%lu order=%d cold=%d
```

These four events deal with page allocation and freeing. mm_page_alloc is a simple indicator of page allocator activity. Pages may be allocated from the per-CPU allocator (high performance) or the buddy allocator.

If pages are allocated directly from the buddy allocator, the mm_page_alloc_zone_locked event is triggered. This event is important as high amounts of activity imply high activity on the zone->lock. Taking this lock impairs performance by disabling interrupts, dirtying cache lines between CPUs and serialising many CPUs.

When a page is freed directly by the caller, the mm_page_free_direct event is triggered. Significant amounts of activity here could indicate that the callers should be batching their activities.

When pages are freed using a pagevec, the mm_pagevec_free is triggered. Broadly speaking, pages are taken off the LRU lock in bulk and freed in batch with a pagevec. Significant amounts of activity here could indicate that the system is under memory pressure and can also indicate contention on the zone->lru_lock.

4. Per-CPU Allocator Activity

```
mm_page_alloc_zone_locked       page=%p pfn=%lu order=%u migratetype=%d cpu=%d
percpu_refill=%d
mm_page_pcpu_drain              page=%p pfn=%lu order=%d cpu=%d migratetype=%d
```

In front of the page allocator is a per-cpu page allocator. It exists only for order-0 pages, reduces contention on the zone->lock and reduces the amount of writing on struct page.

When a per-CPU list is empty or pages of the wrong type are allocated, the zone->lock will be taken once and the per-CPU list refilled. The event triggered is mm_page_alloc_zone_locked for each page allocated with the event indicating whether it is for a percpu_refill or not.

When the per-CPU list is too full, a number of pages are freed, each one which triggers a mm_page_pcpu_drain event.

The individual nature of the events is so that pages can be tracked between allocation and freeing. A number of drain or refill pages that occur consecutively imply the zone->lock being taken once. Large amounts of per-CPU refills and drains could imply an imbalance between CPUs where too much work is being concentrated in one place. It could also indicate that the per-CPU lists should be a larger size. Finally, large amounts of refills on one CPU and drains on another could be a factor in causing large amounts of cache line bounces due to writes between CPUs and worth investigating if pages can be allocated and freed on the same CPU through some algorithm change.

5. External Fragmentation

```
mm_page_alloc_extfrag           page=%p pfn=%lu alloc_order=%d fallback_order=%d
pageblock_order=%d alloc_migratetype=%d fallback_migratetype=%d fragmenting=%d
change_ownership=%d
```

External fragmentation affects whether a high-order allocation will be successful or not. For some types of hardware, this is important although it is avoided where possible. If the system is using huge pages and needs to be able to resize the pool over the lifetime of the system, this value is important.

Large numbers of this event implies that memory is fragmenting and high-order allocations will start failing at some time in the future. One means of reducing the occurrence of this event is to increase the size of

min_free_kbytes in increments of 3*pageblock_size*nr_online_nodes where
pageblock_size is usually the size of the default hugepage size.