How does the SLUB allocator work

Joonsoo Kim
LGE CTO SWP Lab.
iamjoonsoo.kim@lge.com
jsl304@gmail.com
Contents

• Memory allocation hierarchy
• Implementation – the SLUB
• Difference between the SLAB and the SLUB
• Current status
MEMORY ALLOCATION HIERARCHY
Page allocator

- **page allocator**
  - fundamental memory allocator
  - manage all physical memory in system
  - page size = 4096 bytes
  - allocate $2^{\text{order}}$ pages at once

- **limit**
  - size less than page size
What is the SLAB allocator?

• The SLAB allocator
  – in-kernel library like in-userspace library malloc()
  – kmalloc() = malloc()
  – kmem_cache_create(), kmem_cache_alloc(), …

• The object allocator ➔ providing same API
  – The SLAB allocator: traditional allocator
  – The SLOB allocator: for embedded system
  – The SLUB allocator: default, for large system
Allocation hierarchy

- generic kernel code
- object allocator
- page allocator (buddy allocator)
- physical page frame
Warning: term

• “the SLAB allocator” vs “the slab”
  – the SLAB allocator
    • one of the object allocator
  – the slab
    • just data structure
    • used by the slab allocator and the slub allocator
IMPLEMENTATION
- SLUB
Overall structure

UMA, SMP

per node

name
size
align
...

per cpu

cpu0

cpu1

cpu2

cpu3

kmem_cache_cpu

node0

kmem_cache_node

slab caches

slab

slab

slab

slab
Slab

object

fragmentation

index

_mapcount

_lru

...

struct page

freelist

inuse

objects

frozen

_lru

...

struct page

Slab
Per cpu structure

- freelist
- tid
- cpu slab
- partial slabs

kmem_cache_cpu

- slab1
- slab2
- slab3

free objects from slab1

- free object
- slab
Allocation: fast-path

- freelist
- tid
- cpu slab
- partial slabs

kmem_cache_cpu

return this object

free object

slab
Allocation: slow-path

- freelist
- tid
- cpu slab
- partial slabs

kmem_cache_cpu

try on node partial slabs

1

2

3

4

free object

slab
Allocation: very slow-path

**Diagram:**
- **kmem_cache**
- **node0**
- **kmem_cache_node**
- **per node**
  - name
  - size
  - align
  - ...
- **per cpu**
  - cpu0
  - cpu1
  - cpu2
- **freelist**
- **tid**
- **cpu slab**
- **partial slabs**
- **kmem_cache_cpu**
- **slab**

**Process:**
1. Lock
2. Get slab from page allocator
3. Sets lock

**Notes:**
- Allocation is very slow-path.
Free: fast-path

- freelist
- tid
- cpu slab
- partial slabs

kmem_cache_cpu

- slab1
- slab2
- slab3

- object from slab1

- free object
- slab
Free: slow-path

- **case 1.** remote cpu slab
- **case 2.** normal case
- **case 3.** inuse = 0
- **case 4.** first free object

- kmem_cache
  - cpu0
  - cpu1
  - cpu2
  - cpu3
  - slab
- kmem_cache_node
  - node0
  - slab
  - discard
  - slab
  - slab
Performance optimization

- `this_cpu_cmpxchg_double`
  - avoid disabling interrupt
- `cmpxchg_double`
  - avoid taking a lock
Performance optimization: freelist of kmem_cache_cpu

1. Allocation request
2. Read freelist of kmem_cache_cpu
3. Detach first object(A)
4. Assign following object(B) to freelist
5. Return object(A)

Interrupt:
6. Read freelist of kmem_cache_cpu
7. Detach first object(A)
8. Assign following object(B) to freelist
9. Return object(A)

Crash:
- Need irq disable
Performance optimization: freelist of kmem_cache_cpu

allocation request

read freelist, tid of kmem_cache_cpu

detach first object(A)

this_cpu_cmpxchg_double: new freelist(B), new tid

fail

new allocation request

interrupt

read freelist, tid of kmem_cache_cpu

detach first object(A)

this_cpu_cmpxchg_double: new freelist(B), new tid

return object(A)

retry

no need “irq disable”
Performance optimization: freelist of a slab

- acquire freelist of slab
  - read freelist, counter of slab
    - acquire freelist
      - cmpxchg_double_slab: new freelist(NULL), new counter
        - fail
          - retry
  - no need "lock"
    - allocation path

- insert object into freelist of slab
  - read freelist, counter of slab
    - attach one object(A)
      - cmpxchg_double_slab: new freelist(A), new counter
        - success
          - free path
A DIFFERENCE BETWEEN THE SLAB AND THE SLUB
Caching policy

In case of the SLAB

In case of the SLUB
# Free object management

<table>
<thead>
<tr>
<th></th>
<th>cpu object cache</th>
<th>The SLAB</th>
<th>The SLUB</th>
</tr>
</thead>
<tbody>
<tr>
<td>data structure</td>
<td>array</td>
<td>list</td>
<td></td>
</tr>
<tr>
<td>max number of objects</td>
<td>120</td>
<td>don’t care</td>
<td></td>
</tr>
<tr>
<td>size (64bits)</td>
<td>120 * 8 bytes</td>
<td>8 byte</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>slab</th>
<th>The SLAB</th>
<th>The SLUB</th>
</tr>
</thead>
<tbody>
<tr>
<td>data structure</td>
<td>array</td>
<td>list</td>
<td></td>
</tr>
<tr>
<td>max number of objects</td>
<td>202</td>
<td>don’t care</td>
<td></td>
</tr>
<tr>
<td>size (64bits)</td>
<td>202 * 4 bytes</td>
<td>8 byte</td>
<td>(overload “struct page”)</td>
</tr>
</tbody>
</table>
Miscellaneous

- kmalloc alignment
- fallback order slab
- kmem_cache alignment
- debugging feature
- NUMA
CURRENT STATUS
Trends

- per cpu partial lists
- common sl[aou]b
- slab accounting for memcg
Any questions?