Zoned Allocator -6- (Watermark)

2016-06-29 (http://jake.dothome.co.kr/zonned-allocator-watermark/) Moon Young-il (http://jake.dothome.co.kr/author/admin/) Linux Kernel (http://jake.dothome.co.kr/category/linux/)

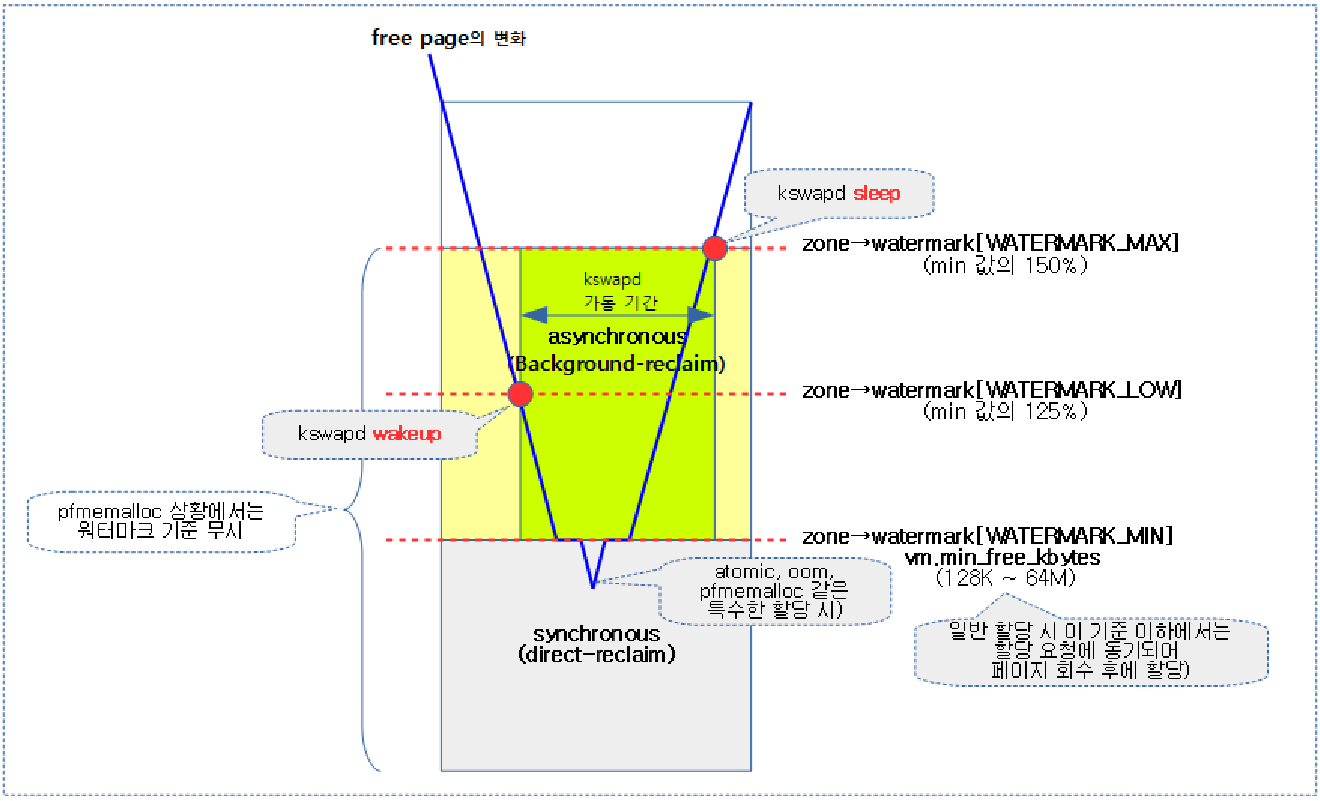
<kernel v5.0>

Zone Watermark

When attempting to allocate a page in a zone, the remaining free pages are compared with the watermark criteria for each zone to determine whether memory is allocated. These watermarks are used by specifying three watermark reference values as follows, and each of them has the following meanings.

- WATERMARK_MIN
  - It is equal to the lowest reference point of low memory.
  - vm.min_free_kbytes has the number of pages corresponding to the value.
  - If the calculated free page falls below this value, it is not possible to retrieve the page. In this case, if you allow a synchronous direct page reclamation operation, the allocation continues after the page reclaim.
  - Under special circumstances (with the exception of atomic, oom, pfmemalloc), page allocation is possible even below this standard.
- WATERMARK_LOW
  - By default, the number of pages has a value of 125% of the min value. This is the point at which kswapd automatically wakes up.
    - Users can change their watermark_scale_factor to raise this bar even further.
- WATERMARK_MAX
  - This is the number of pages with a value of 150% of the min value by default, and this is the point at which kswapd is automatically sleeped.
  - Users can change their watermark_scale_factor to raise this bar even further.

The following figure shows the relationship between allocation and page recall based on zone watermark settings.



(http://jake.dothome.co.kr/wp-content/uploads/2016/06/zone-watermark-1c.png)

You can check the nr_free_pages and min, low, and high watermark values as follows.

```
$ cat /proc/zoneinfo
Node 0, zone    DMA32
  per-node stats
(...생략...)
  pages free     633680
        min      5632             <-----
        low      7040             <-----
        high     8448             <-----
        spanned  786432
        present  786432
        managed  765771
        protection: (0, 0, 0)
      nr_free_pages 633680        <-----
(...생략...)
```

managed_pages

The number of pages that the buddy system manages for each zone and the initial values in the DMA and normal zones are as follows:

- managed_pages = present_pages – the number of pages that were reserved just before the buddy system started
- managed_pages = spanned_pages – absent_pages – (memmap_pages + dma_reserve + …)
  - present_pages = spanned_pages – absent_pages

You can check the managed_pages values for each zone as follows.

```
$ cat /proc/zoneinfo
Node 0, zone    DMA32
  per-node stats
(...생략...)
  pages free     633680
        min      5632
        low      7040
        high     8448
        spanned  786432
        present  786432
        managed  765771           <-----
        protection: (0, 0, 0)
      nr_free_pages 633680
(...생략...)
```

Quick Watermark Comparison

zone_watermark_fast()

mm/page_alloc.c

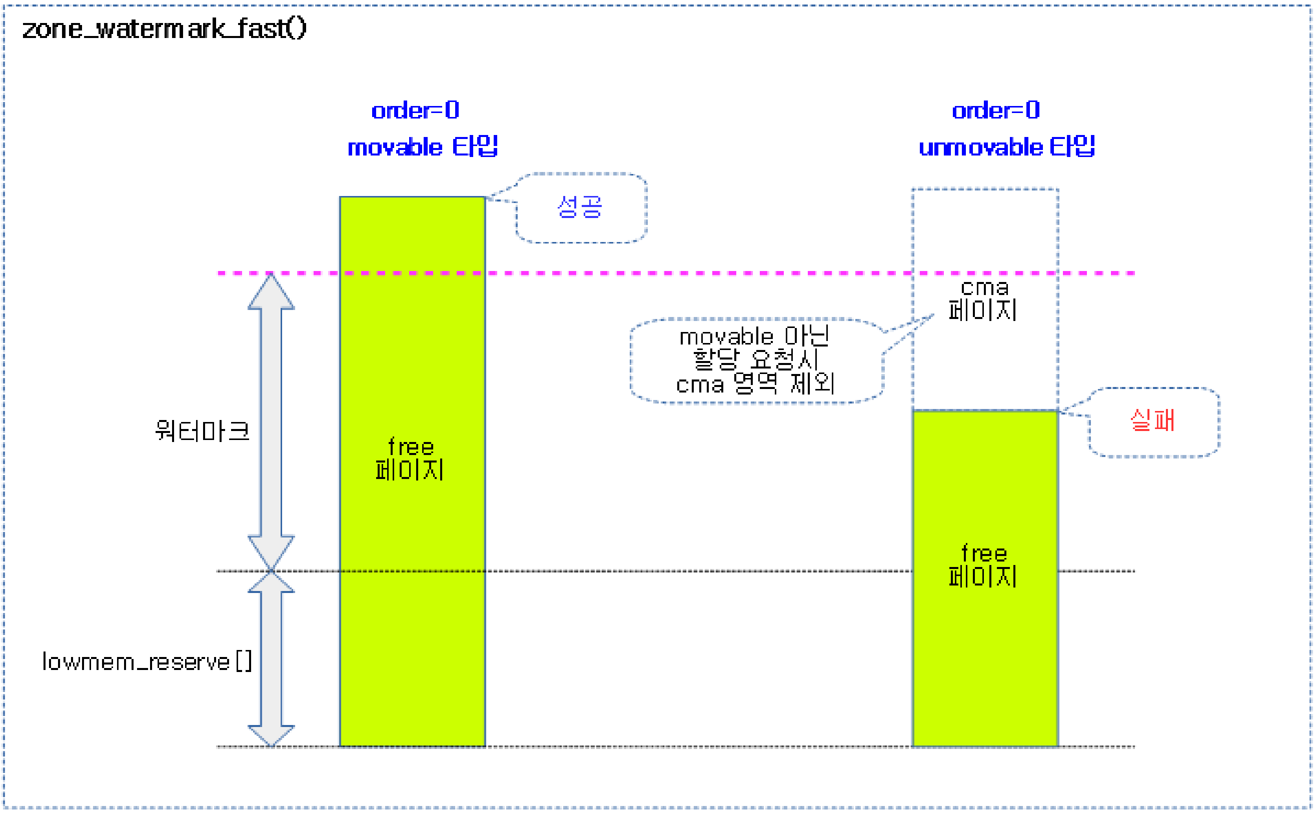
```
01 static inline bool zone_watermark_fast(struct zone *z, unsigned int order,
02                 unsigned long mark, int classzone_idx, unsigned int alloc_flags)
03 {
04         long free_pages = zone_page_state(z, NR_FREE_PAGES);
05         long cma_pages = 0;
06 
07 #ifdef CONFIG_CMA
08         /* If allocation can't use CMA areas don't use free CMA pages */
09         if (!(alloc_flags & ALLOC_CMA))
10                 cma_pages = zone_page_state(z, NR_FREE_CMA_PAGES);
11 #endif
12 
13         /*
14          * Fast check for order-0 only. If this fails then the reserves
15          * need to be calculated. There is a corner case where the check
16          * passes but only the high-order atomic reserve are free. If
17          * the caller is !atomic then it'll uselessly search the free
18          * list. That corner case is then slower but it is harmless.
19          */
20         if (!order && (free_pages - cma_pages) > mark + z->lowmem_reserve[classzone_idx])
21                 return true;
22 
23         return __zone_watermark_ok(z, order, mark, classzone_idx, alloc_flags,
24                                         free_pages);
25 }
```

It quickly determines whether the requested @z zone can secure @order pages from a free page that is higher than the @mark standard. Judgment logic has been added exclusively for order 0 allocation requests. (true=OK above the watermark standard, false=NOT-OK below the watermark standard)

- In line 4 of code, we find the free page for that zone.

- If the page request is not movable in line 9~10 of the code, the pages in the cma area will not be available. First, we know how many pages are using the CMA area.
- In line 19~20 of the code, if the request is to assign order 0, you can make a quick judgment. Returns true if the free page, except for the CMA page, exceeds the watermark threshold plus the lowmem reserve area.
- In line 22~23 of the code, order returns whether the allocation request of 1 or higher exceeds the watermark criterion.

The following figure shows how the order 0 page allocation request is judged by the watermark criterion for quick assignment.



(http://jake.dothome.co.kr/wp-content/uploads/2016/06/zone_watermark_fast-1a.png)

Comparison of Default Watermarks

zone_watermark_ok()

mm/page_alloc.c

```
bool zone_watermark_ok(struct zone *z, unsigned int order, unsigned long mark,
                      int classzone_idx, int alloc_flags)
{
        return __zone_watermark_ok(z, order, mark, classzone_idx, alloc_flags,
                                        zone_page_state(z, NR_FREE_PAGES));
}
```

In order to handle the order page in that zone, make sure that the free page is sufficient for the watermark value.

__zone_watermark_ok()

mm/page_alloc.c

```
/*
 * Return true if free base pages are above 'mark'. For high-order checks it
 * will return true of the order-0 watermark is reached and there is at least
 * one free page of a suitable size. Checking now avoids taking the zone lock
 * to check in the allocation paths if no pages are free.
 */
```

```
bool __zone_watermark_ok(struct zone *z, unsigned int order, unsigned long mark,
                         int classzone_idx, unsigned int alloc_flags,
                         long free_pages)
{
        long min = mark;
        int o;
        const bool alloc_harder = (alloc_flags & (ALLOC_HARDER|ALLOC_OOM));

        /* free_pages may go negative - that's OK */
        free_pages -= (1 << order) - 1;

        if (alloc_flags & ALLOC_HIGH)
                min -= min / 2;

        /*
         * If the caller does not have rights to ALLOC_HARDER then subtract
         * the high-atomic reserves. This will over-estimate the size of the
         * atomic reserve but it avoids a search.
         */
        if (likely(!alloc_harder)) {
                free_pages -= z->nr_reserved_highatomic;
        } else {
                /*
                 * OOM victims can try even harder than normal ALLOC_HARDER
                 * users on the grounds that it's definitely going to be in
                 * the exit path shortly and free memory. Any allocation it
                 * makes during the free path will be small and short-lived.
                 */
                if (alloc_flags & ALLOC_OOM)
                        min -= min / 2;
                else
                        min -= min / 4;
        }


#ifdef CONFIG_CMA
        /* If allocation can't use CMA areas don't use free CMA pages */
        if (!(alloc_flags & ALLOC_CMA))
                free_pages -= zone_page_state(z, NR_FREE_CMA_PAGES);
#endif
```

```
41
42          /*
43           * Check watermarks for an order-0 allocation request. If these
44           * are not met, then a high-order request also cannot go ahead
45           * even if a suitable page happened to be free.
46           */
47          if (free_pages <= min + z->lowmem_reserve[classzone_idx])
48                  return false;
49
50          /* If this is an order-0 request then the watermark is fine */
51          if (!order)
52                  return true;
53
54          /* For a high-order request, check at least one suitable page is free */
55          for (o = order; o < MAX_ORDER; o++) {
56                  struct free_area *area = &z->free_area[o];
57                  int mt;
58
59                  if (!area->nr_free)
60                          continue;
61
62                  for (mt = 0; mt < MIGRATE_PCPTYPES; mt++) {
63                          if (!list_empty(&area->free_list[mt]))
64                                  return true;
65                  }
66
67  #ifdef CONFIG_CMA
68                  if ((alloc_flags & ALLOC_CMA) &&
69                      !list_empty(&area->free_list[MIGRATE_CMA])) {
70                          return true;
71                  }
72  #endif
73                  if (alloc_harder &&
74                          !list_empty(&area->free_list[MIGRATE_HIGHATOMIC]))
75                          return true;
76          }
77          return false;
78  }
```

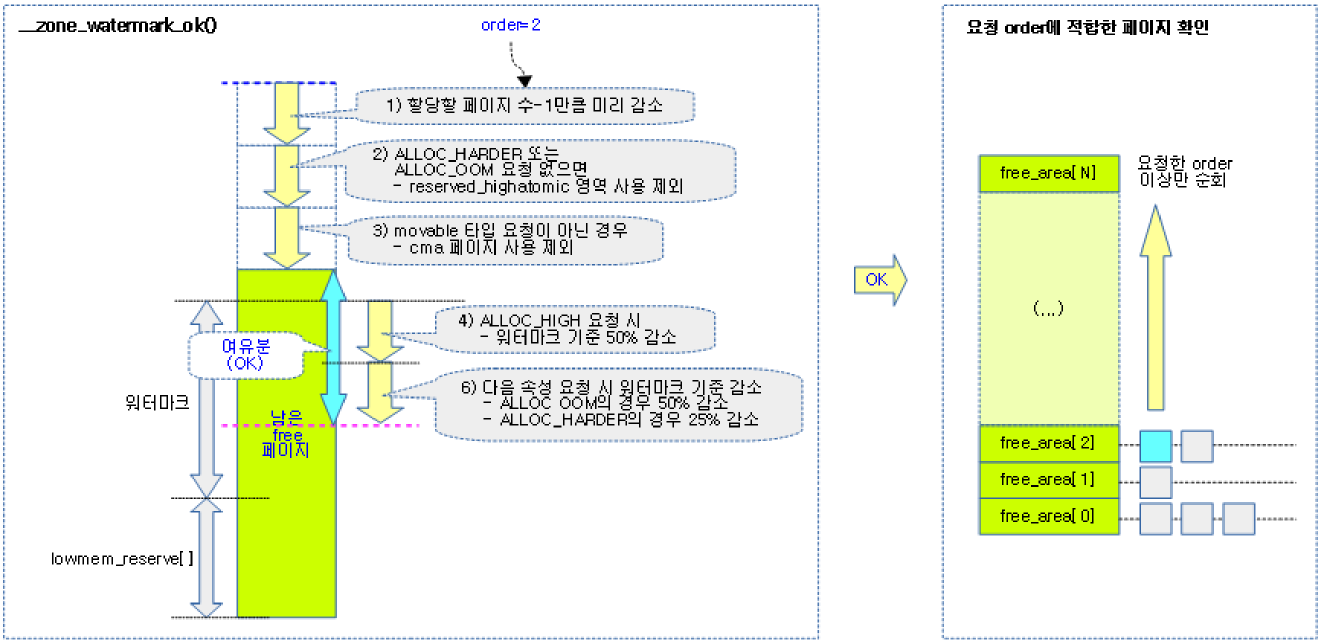
In order to handle the order page in that zone, make sure that the free page is sufficient for the watermark value.

- In line 5 of the code, assign the watermark value to the min value.
- In line 7 of the code, it is decided whether the watermark standard should be lowered as much as possible to determine whether the allocation should be made in the alloc_harder.
  - Allocation in gfp_atomic requests and OOM situations requires the system to complete the allocation as much as possible.
- In line 10 of code, the number of pages requested to allocate free pages is reduced by -1.
- In line 12~13 of the code, the ALLOC_HIGH assignment flag is set in the case of an atomic request, and the watermark criteria are halved to allow for maximum allocation.
- Unless it is alloc_harder in line 20~21 of the code, the reserved_highatomic pages are omitted to prevent use.
- In line 22~33 of the code, the watermark is lowered further in the case of alloc_harder situation, which is 25% lower in the case of an atomic request and 50% in the case of an OOM situation.
- If the page allocation request is not movable on lines 38~39 of the code, the cma area cannot be used. Therefore, the pages in the CMA area are excluded from the free page count first.

- In line 47~48 of the code, if the free page is below the watermark threshold and is low on memory, including the lowmem reserve area, it will return false.
- On lines 51~52 of the code, it returns true for an order 0 assignment request.
- In line 55~76 of the code, for requests with order 1 or more, the actual free list is searched and returned after checking whether it is assignable. [request order, MAX_ORDER) traverses the free_area in the range, and returns true because it is assignable if it is included in the following 3 criteria:
  - If there is a free page in the list of type unmovable, movable, or reclaimable.
  - If the request is a movable page that can use the CMA area, if there is a free page in the list of type CMA.
  - In alloc_harder situation, if there is a free page in a list of type highatomic

The following figure shows checking whether a free page can be assigned according to the zone watermark setting.

- After checking whether there is enough space for the remaining free pages as shown in the figure on the left,
- Search for free_area[], as shown in the figure on the right, to see if it is finally assignable.
- The zone index used in lowmem_reserve[] is the preferred zone index that was initially requested.



(http://jake.dothome.co.kr/wp-content/uploads/2016/06/zone_watermark_ok-1.png)

Accurate Watermark Comparison

zone_watermark_ok_safe()

page_alloc.c

```
bool zone_watermark_ok_safe(struct zone *z, unsigned int order,
                                unsigned long mark, int classzone_idx)
{
        long free_pages = zone_page_state(z, NR_FREE_PAGES);
```

```
05
06         if (z->percpu_drift_mark && free_pages < z->percpu_drift_mark)
07                 free_pages = zone_page_state_snapshot(z, NR_FREE_PAGES);
08
09         return __zone_watermark_ok(z, order, mark, classzone_idx, 0,
10                                                                 free_pages);
11 }
```

In order to process the order page in the zone, make sure that the correctly calculated free page is sufficient for the watermark value.

- In line 4 of the code, we first get the approximate number of free pages.
- If the number of free pages from code lines 6~7 is less than percpu_drift_mark, the free pages are calculated correctly.
  - percpu_drift_mark value is set in the refresh_zone_stat_thresholds() function.
- In line 9~10 of the code, compare the number of free pages with the watermark and return whether it can be assigned or not.

Zone Counter (STAT)

The zone counters (stats) have two separate zone counters and two per-CPU counters for performance. If only an approximate value of the zone is required, the zone counter value is returned as it is for performance, and the zone counter and per-cpu counter are added only if the correct calculation is required.

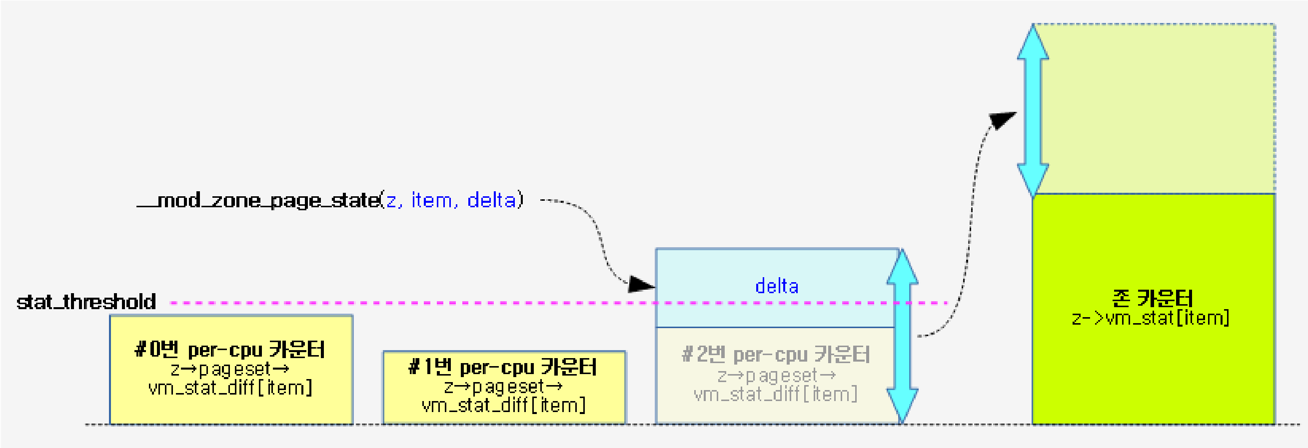
percpu_drift_mark

In kernel memory management, a routine is often used to determine memory shortage by reading the number of free pages remaining and comparing it with the watermark. However, in order to read the correct free page value, both the zone counter and the per-cpu counter must be read and added, and this calculation decreases the performance, so the performance is maintained by reading only the zone counter value and comparing it with the percpu_drift_mark value set to a certain size larger than the high watermark and inducing more accurate calculations only when it is below this value.

stat_threshold

When the counter increases or decreases, the per-CPU counter is increased or decreased first for performance. For the accuracy of the zone counter, the value of the per-cpu counter is transferred to the zone counter and added only when the value exceeds a certain standard stat_threshold.

The following figure shows how when the zone counter of the CPU changes, it is moved to the zone counter if it exceeds stat_threshold.



(http://jake.dothome.co.kr/wp-content/uploads/2016/06/mod_zone_page_state-1a.png)

You can check the stat_threshold values as follows:

```
$ cat /proc/zoneinfo
Node 0, zone    DMA32
  per-node stats
(...생략...)
  pagesets
    cpu: 0
              count: 143
              high:  378
              batch: 63
  vm stats threshold: 24        <-----
    cpu: 1
              count: 285
              high:  378
              batch: 63
  vm stats threshold: 24        <-----
(...생략...)
```

zone_page_state()

include/linux/vmstat.h

```
static inline unsigned long zone_page_state(struct zone *zone,
                                        enum zone_stat_item item)
{
        long x = atomic_long_read(&zone->vm_stat[item]);
#ifdef CONFIG_SMP
        if (x < 0)
                x = 0;
#endif
        return x;
}
```

Returns the approximate VM stat value for the zone's @item.

- Returns only the vm stat value of the zone.

zone_page_state_snapshot()

include/linux/vmstat.h

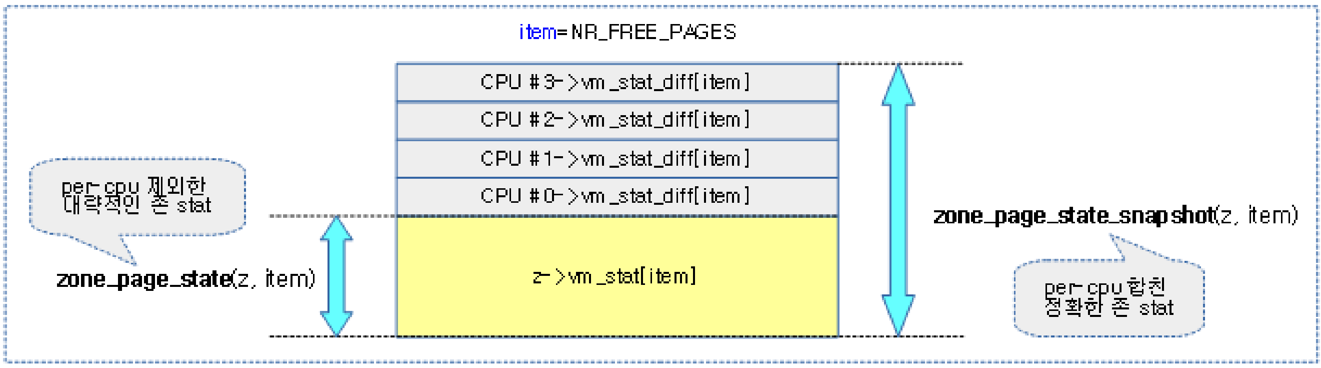
```
/*
 * More accurate version that also considers the currently pending
 * deltas. For that we need to loop over all cpus to find the current
 * deltas. There is no synchronization so the result cannot be
 * exactly accurate either.
 */
```

```
static inline unsigned long zone_page_state_snapshot(struct zone *zone,
                                        enum zone_stat_item item)
{
        long x = atomic_long_read(&zone->vm_stat[item]);

#ifdef CONFIG_SMP
        int cpu;
        for_each_online_cpu(cpu)
                x += per_cpu_ptr(zone->pageset, cpu)->vm_stat_diff[item];

        if (x < 0)
                x = 0;
#endif
        return x;
}
```

Accurately calculates the vm stat value corresponding to the zone's @item.

- Use the following formula to return the exact value:
    - vm stat value in zone + all per-cpu vm stat value

The following figure shows the difference between the zone_page_state_snapshop() function and the zone_page_state() function.

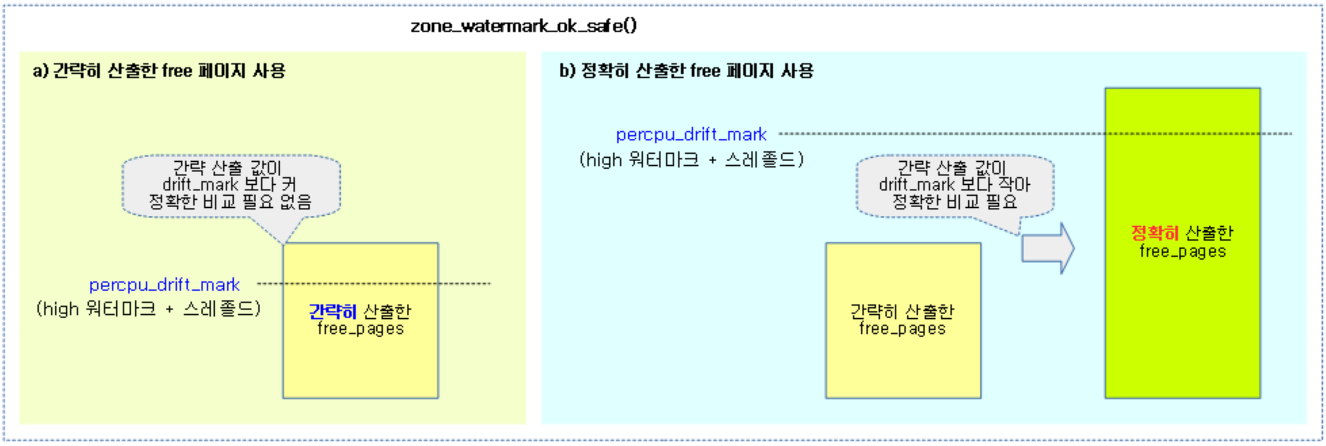


(http://jake.dothome.co.kr/wp-content/uploads/2016/06/zone_page_state_snapshot-1a.png)

What's the difference between the zone_watermark_ok() function and the zone_watermark_ok_safe() function?

- zone_watermark_ok() function
    - Use the free page value, which is roughly calculated for the number of free pages to compare with the watermark.
- zone_watermark_ok_safe() function
    - Use the free page value calculated exactly for the number of free pages to be compared with the watermark.

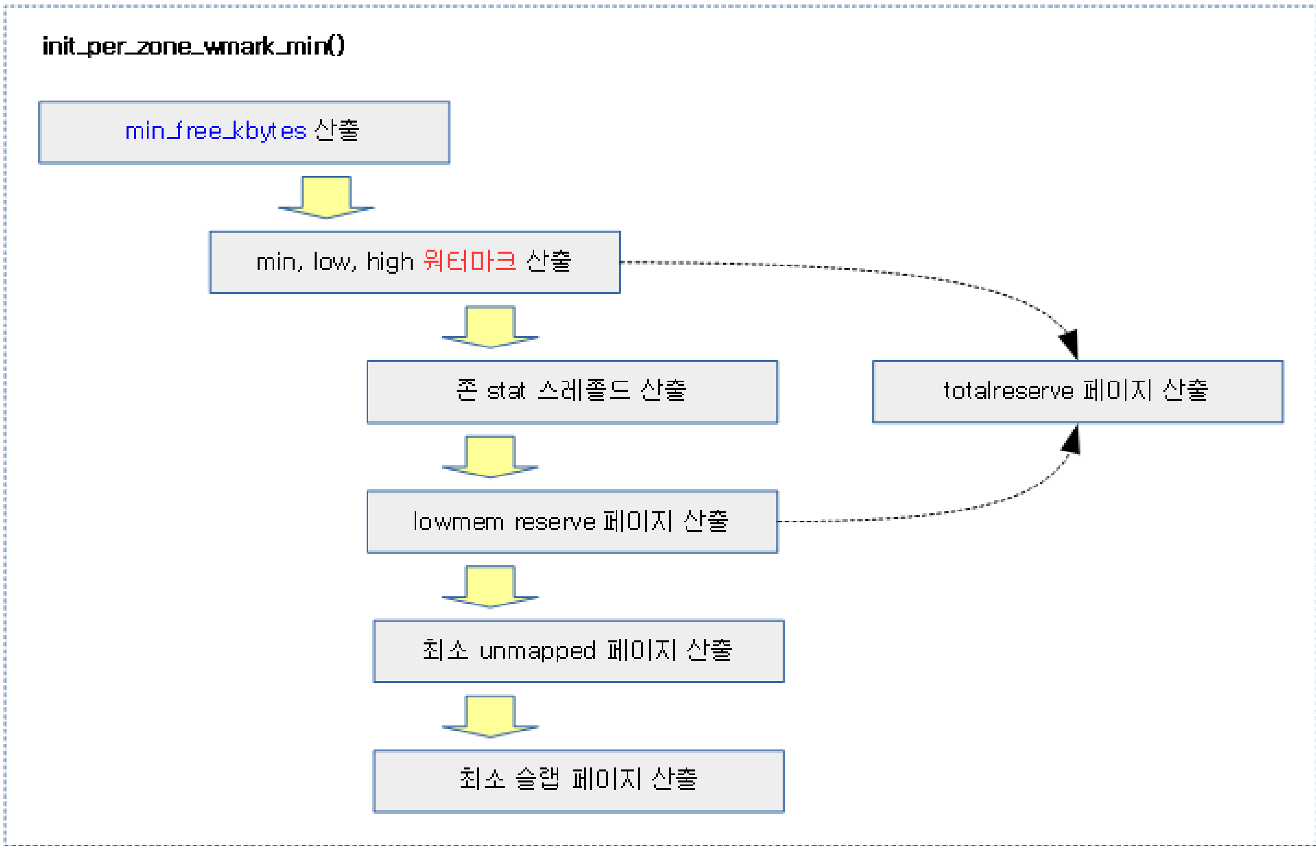
Below is an example of the use of the zone_watermark_ok_safe() function using the zone stat thread.



(http://jake.dothome.co.kr/wp-content/uploads/2016/06/zone_watermark_ok_safe-1a.png)

Reset watermark and related settings

Resetting the watermark and related settings is done as follows.



(http://jake.dothome.co.kr/wp-content/uploads/2016/06/init_per_zone_wmark_min-2a.png)

Set the watermark min value

init_per_zone_wmark_min()

mm/page_alloc.c

```
/*
 * Initialise min_free_kbytes.
 *
 * For small machines we want it small (128k min).  For large machines
 * we want it large (64MB max).  But it is not linear, because network
 * bandwidth does not increase linearly with machine size.  We use
 *
 *      min_free_kbytes = 4 * sqrt(lowmem_kbytes), for better accuracy:
 *      min_free_kbytes = sqrt(lowmem_kbytes * 16)
 *
 * which yields
 *
 * 16MB:        512k
 * 32MB:        724k
 * 64MB:        1024k
 * 128MB:       1448k
 * 256MB:       2048k
 * 512MB:       2896k
 * 1024MB:      4096k
 * 2048MB:      5792k
 * 4096MB:      8192k
 * 8192MB:      11584k
 * 16384MB:     16384k
 */
```

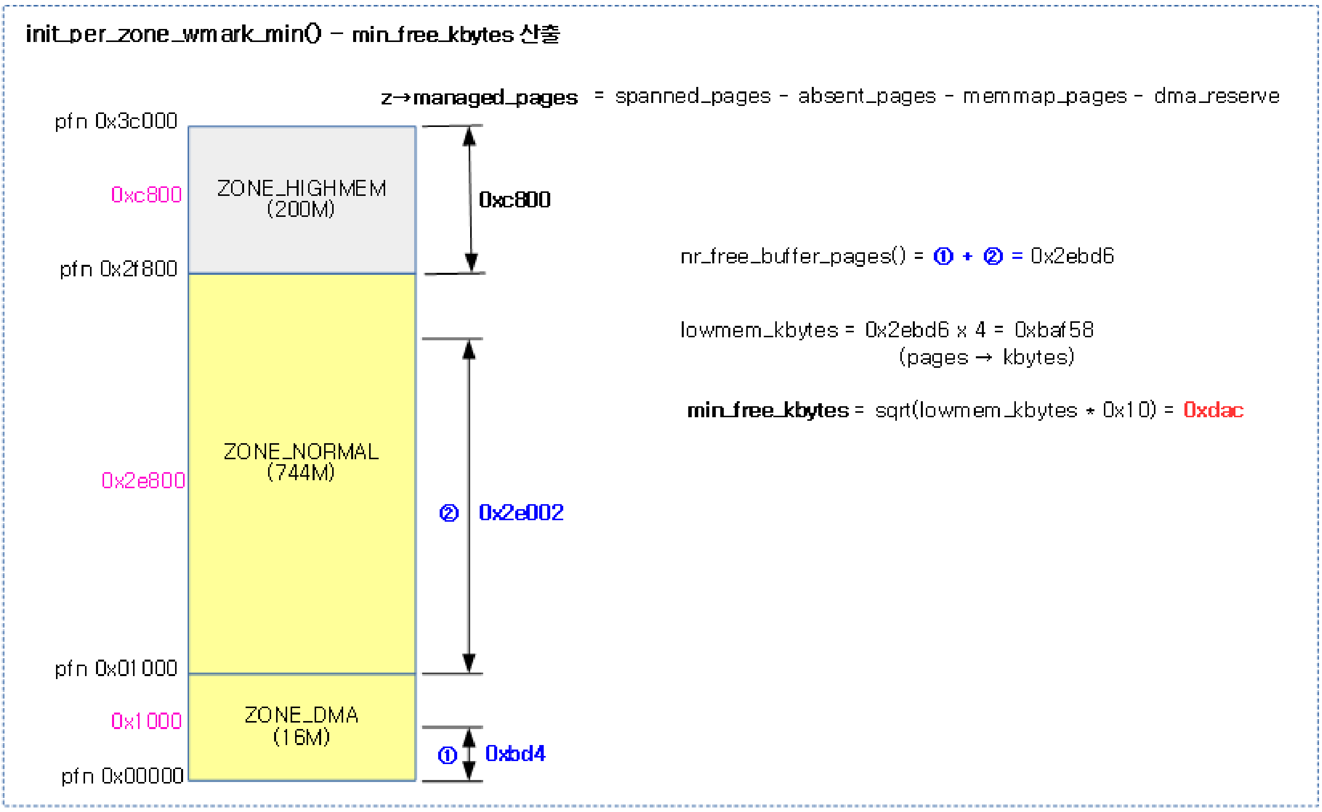
```
int __meminit init_per_zone_wmark_min(void)
{
        unsigned long lowmem_kbytes;
        int new_min_free_kbytes;

        lowmem_kbytes = nr_free_buffer_pages() * (PAGE_SIZE >> 10);
        new_min_free_kbytes = int_sqrt(lowmem_kbytes * 16);

        if (new_min_free_kbytes > user_min_free_kbytes) {
                min_free_kbytes = new_min_free_kbytes;
                if (min_free_kbytes < 128)
                        min_free_kbytes = 128;
                if (min_free_kbytes > 65536)
                        min_free_kbytes = 65536;
        } else {
                pr_warn("min_free_kbytes is not updated to %d because user defined value %d is prefee
rred\n",
                                new_min_free_kbytes, user_min_free_kbytes);
        }
        setup_per_zone_wmarks();
        refresh_zone_stat_thresholds();
        setup_per_zone_lowmem_reserve();

#ifdef CONFIG_NUMA
        setup_min_unmapped_ratio();
        setup_min_slab_ratio();
#endif

        return 0;
}
core_initcall(init_per_zone_wmark_min)
```

It uses lowmem's available page size to calculate the min_free_kbytes value used as the basis for the watermark. In addition, it calculates watermarks, lowmem_reserve, totalreserve, min umpapped, min slap pages, etc. for each zone.

- In line 6~18 of the code, the number of available pages in lowmem is used to calculate the min_free_kbytes value used as the watermark reference value.
  - Lowmem Available Page Calculation Formula
    - managed_pages – High Watermark Page
  - min_free_kbytes Calculation Formula
    - sqrt((Number of available pages in lowmem >> 10) * 16)
      - 예) sqrt(512M * 0x10) -> 2M
    - Calculated values are limited to the range of 128 ~ 65536 (range of 128K ~ 64M)
  - In addition to initializing the kernel, this initialization routine is performed whenever the memory is hot-add/del using hotplug memory.
  - When the kernel is initialized for the first time, the initial value of each zone watermark is 0.
- In line 19 of the code, it calculates the watermark value and totalreserve_pages value for each zone.
- On line 20 of code, we calculate the stat thread and percpu_drift_mark for each zone.
- Line 21 of the code yields the zone->lowmem_reserve and totalreserve pages.
- Yield at least ummapped pages at line 24 of code.
- Yield a minimum of slab pages on line 25 of code.

The following figure shows how the global min_free_kbytes value is calculated.

- It is calculated by multiplying the available pages (managed – high watermark) of lowmem (dma, normal) by 16 and root to the square.
  - On the first bootup, the high watermark value is 0.
- In the case of ARM64, the dma32 zone is used, and the highmem zone is not used.



(http://jake.dothome.co.kr/wp-content/uploads/2016/06/init_per_zone_wmark_min-1a.png)

Calculate the number of lowmem available pages

nr_free_buffer_pages()

mm/page_alloc.c

```
/**
 * nr_free_buffer_pages - count number of pages beyond high watermark
 *
 * nr_free_buffer_pages() counts the number of pages which are beyond th
e high
 * watermark within ZONE_DMA and ZONE_NORMAL.
 */
```

```
unsigned long nr_free_buffer_pages(void)
{
        return nr_free_zone_pages(gfp_zone(GFP_USER));
}
EXPORT_SYMBOL_GPL(nr_free_buffer_pages);
```

Returns the number of available pages in the current node's zonelist for zones below the normal zone.

- Note that the first time it is called to configure the kernel, the high watermark is 0.
- Number of pages available
  - managed_pages – High Watermark Page

nr_free_zone_pages()

mm/page_alloc.c

```
/**
 * nr_free_zone_pages - count number of pages beyond high watermark
 * @offset: The zone index of the highest zone
 *
 * nr_free_zone_pages() counts the number of counts pages which are beyo
nd the
 * high watermark within all zones at or below a given zone index.  For
each
 * zone, the number of pages is calculated as:
 *
 *     nr_free_zone_pages = managed_pages - high_pages
 */
```

```
static unsigned long nr_free_zone_pages(int offset)
{
        struct zoneref *z;
        struct zone *zone;

        /* Just pick one node, since fallback list is circular */
        unsigned long sum = 0;

        struct zonelist *zonelist = node_zonelist(numa_node_id(), GFP_KE
RNEL);

        for_each_zone_zonelist(zone, z, zonelist, offset) {
                unsigned long size = zone_managed_pages(zone);
                unsigned long high = high_wmark_pages(zone);
                if (size > high)
                        sum += size - high;
        }

```

```
        return sum;
}
```

Returns the current node's zonelist plus all available pages for zones offset or lower.

- Number of pages available
    - managed_pages – High Watermark Page

Watermark Ratio Settings

The spacing between watermarks uses 25% of the watermark min value by default, and users can change the watermark ratio to increase kswapd efficiency.

watermark_scale_factor

/proc/sys/vm/watermark_scale_factor and has an initial value of 10. When the initial value is 10, it means that it is the value of 0.1% of the memory, and when the maximum value is 1000, it is the value of 10% of the memory. In this case, the maximum interval between min and low and low and high is possible up to 10% of the memory (managed_pages).

- In kernel v4.6-rc1, the threshold of the watermark low and high values has been added to increase the efficiency of kswapd on systems with large memory.

---

min, low, high watermark output

Using min_free_kbytes values determined by the amount of memory, min, low, and high watermarks are calculated for each zone.

setup_per_zone_wmarks()

mm/page_alloc.c

```
/**
 * setup_per_zone_wmarks - called when min_free_kbytes changes
 * or when memory is hot-{added|removed}
 *
 * Ensures that the watermark[min,low,high] values for each zone are set
 * correctly with respect to min_free_kbytes.
 */
```

```
void setup_per_zone_wmarks(void)
{
        mutex_lock(&zonelists_mutex);
        __setup_per_zone_wmarks();
        mutex_unlock(&zonelists_mutex);
}
```

It calculates min, low, and high watermarks for each zone.

__setup_per_zone_wmarks()

mm/page_alloc.c

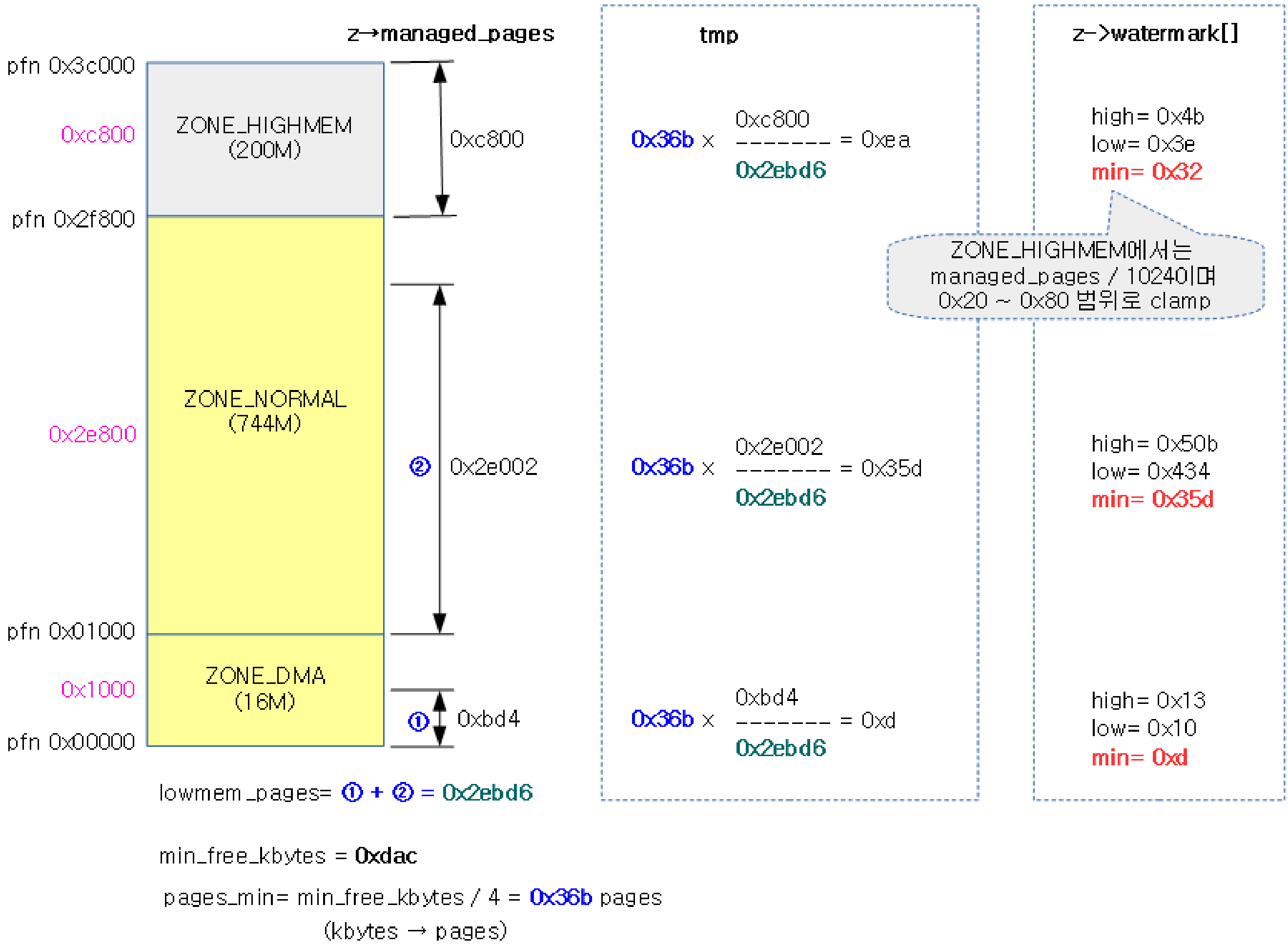
```
static void __setup_per_zone_wmarks(void)
{
        unsigned long pages_min = min_free_kbytes >> (PAGE_SHIFT - 10);
        unsigned long lowmem_pages = 0;
        struct zone *zone;
        unsigned long flags;

        /* Calculate total number of !ZONE_HIGHMEM pages */
        for_each_zone(zone) {
                if (!is_highmem(zone))
                        lowmem_pages += zone_managed_pages(zone);
        }

        for_each_zone(zone) {
                u64 tmp;

                spin_lock_irqsave(&zone->lock, flags);
                tmp = (u64)pages_min * zone_managed_pages(zone);
                do_div(tmp, lowmem_pages);
                if (is_highmem(zone)) {
                        /*
                         * __GFP_HIGH and PF_MEMALLOC allocations usually don't
                         * need highmem pages, so cap pages_min to a small
                         * value here.
                         *
                         * The WMARK_HIGH-WMARK_LOW and (WMARK_LOW-WMARK_MIN)
                         * deltas control asynch page reclaim, and so should
                         * not be capped for highmem.
                         */
                        unsigned long min_pages;

                        min_pages = zone_managed_pages(zone) / 1024;
                        min_pages = clamp(min_pages, SWAP_CLUSTER_MAX, 128UL);
                        zone->_watermark[WMARK_MIN] = min_pages;
                } else {
                        /*
                         * If it's a lowmem zone, reserve a number of pages
                         * proportionate to the zone's size.
                         */
                        zone->_watermark[WMARK_MIN] = tmp;
                }

                /*
                 * Set the kswapd watermarks distance according to the
                 * scale factor in proportion to available memory, but
                 * ensure a minimum size on small systems.
                 */
                tmp = max_t(u64, tmp >> 2,
                            mult_frac(zone_managed_pages(zone),
                                      watermark_scale_factor, 10000));

                zone->_watermark[WMARK_LOW]  = min_wmark_pages(zone) + tmp;
                zone->_watermark[WMARK_HIGH] = min_wmark_pages(zone) + tmp * 2;
                zone->watermark_boost = 0;

                spin_unlock_irqrestore(&zone->lock, flags);
```

```
57         }
58 
59         /* update totalreserve_pages */
60         calculate_totalreserve_pages();
61 }
```

Calculate low, min, and high watermarks for each zone and update the TotalReserve page.

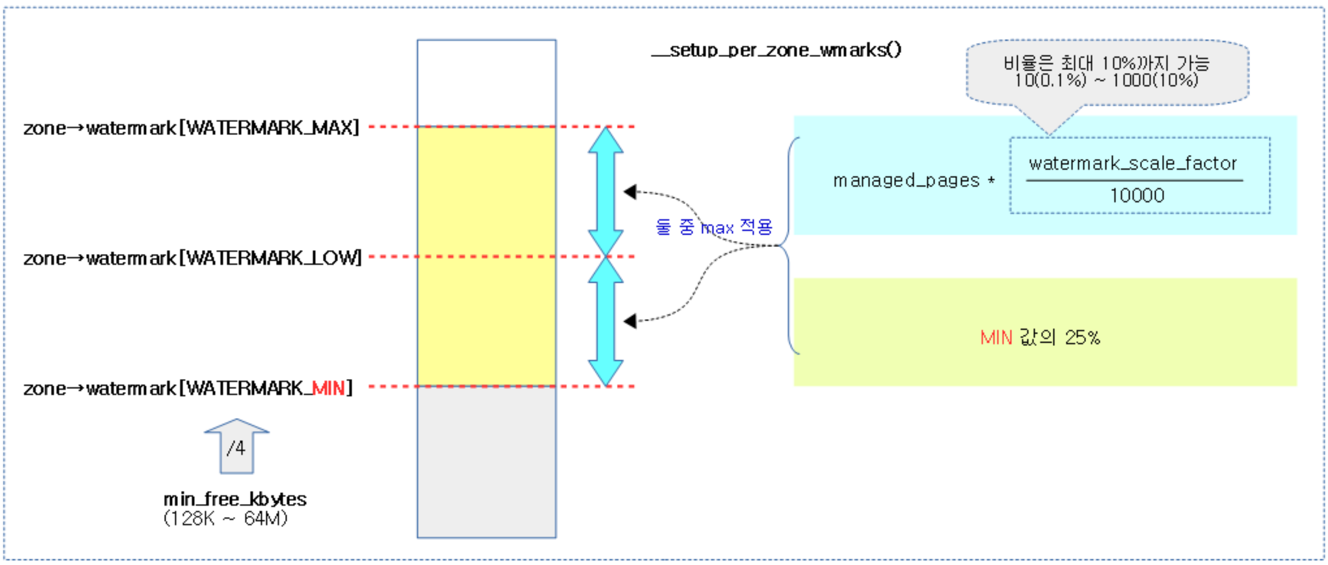
- In line 3 of code, replace the global min_free_kbytes value in terms of pages and assign it to the pages_min.
- In line 9~12 of the code, sum the managed_pages of the lowmem area to get the lowmem_pages.
- In code lines 14~19, go through each zone and set the pages_min (min_free_kbytes to pages) to the number of pages equal to the percentage of the current zone in the TMP.
- In line 20~41 of the code, if it is a highmem zone, the actual managed_pages divided by 1024 is limited to the range of 32~128 and the value is stored in the min watermark, and if it is not a highmem zone, the tmp value calculated above is set in the min watermark.
- In code lines 48~53, the spacing between each watermark is determined by the greater of the following two calculated values.
  - 25% of the initial calculated min value
  - managed_pages * Watermark Ratio (0.1 ~ 10%)
- In line 54 of code, reset the Watermark Boost value to 0.
- Since the high watermark value has been updated in line 60 of the code, it is updated by calculating the number of pages in totalreserve.
  - Whenever the high watermark and lowmem reserve pages are recalfigured, the totalreserve value is also updated.

The following figure shows the min_free_kbytes value and the available pages (managed_pages – high watermarks) in the lowmem area and the min, low, and high watermarks in the proportions of each zone.



(http://jake.dothome.co.kr/wp-content/uploads/2016/06/setup_per_zone_wmarks-1.png)

The following figure shows the largest of the two calculated values on the right as the watermark spacing value.



(http://jake.dothome.co.kr/wp-content/uploads/2016/06/setup_per_zone_wmarks-2.png)

Zone stat threaded output

High watermark to calculate stat threads for zones and nodes. The calculated items are as follows:

- node stat threaded
- John Stat Srezold
- percpu_drift_mark

refresh_zone_stat_thresholds()

mm/vmstat.c

```
/*
 * Refresh the thresholds for each zone.
 */
```

```
void refresh_zone_stat_thresholds(void)
{
        struct pglist_data *pgdat;
        struct zone *zone;
        int cpu;
        int threshold;

        /* Zero current pgdat thresholds */
        for_each_online_pgdat(pgdat) {
                for_each_online_cpu(cpu) {
                        per_cpu_ptr(pgdat->per_cpu_nodestats, cpu)->stat_threshold = 0;
                }
        }

        for_each_populated_zone(zone) {
                struct pglist_data *pgdat = zone->zone_pgdat;
                unsigned long max_drift, tolerate_drift;

                threshold = calculate_normal_threshold(zone);

                for_each_online_cpu(cpu) {
                        int pgdat_threshold;

                        per_cpu_ptr(zone->pageset, cpu)->stat_threshold
                                                        = threshold;

                        /* Base nodestat threshold on the largest populated zone. */
                        pgdat_threshold = per_cpu_ptr(pgdat->per_cpu_nodestats, cpu)->stat_thresholdd
;
                        per_cpu_ptr(pgdat->per_cpu_nodestats, cpu)->stat_threshold
                                = max(threshold, pgdat_threshold);
                }

                /*
                 * Only set percpu_drift_mark if there is a danger that
                 * NR_FREE_PAGES reports the low watermark is ok when in fact
                 * the min watermark could be breached by an allocation
                 */
                tolerate_drift = low_wmark_pages(zone) - min_wmark_pages(zone);
                max_drift = num_online_cpus() * threshold;
                if (max_drift > tolerate_drift)
```

```
42                              zone->percpu_drift_mark = high_wmark_pages(zone) +
43                                              max_drift;
44          }
45  }
```

For all nodes and zones, it calculates node stat threads, zone stat threads, and percpu_drift_mark.

- In line 9~13 of the code, traverse the entire node and initialize the node stat thread to 0.
- Traversing the populate zone in code lines 15~25 and yielding the zone stat thread value.
- On lines 28~31 of the code, update the node stat thread with the largest zone stat thread value.
- In lines 38~42 of the code, we calculate the percpu_drift_mark that is the zone member.
    - max_drift = Number of online CPUs * Calculated thread value
    - If the max_drift value is greater than the spacing between the watermarks, use a high watermark + max_drift value.

a) Calculating threads to be used for memory decompression

The normal thread value is set by being called and calculated in the following cases.

- The refresh_zone_stat_thresholds() function that is called as the memory changes due to hotplugs and hotremovs in the bootup and memory
- Just before kswapd goes to sleep in the kswapd_try_to_sleep() function

calculate_normal_threshold()

mm/vmstat.c

```
int calculate_normal_threshold(struct zone *zone)
{
        int threshold;
        int mem;        /* memory in 128 MB units */

        /*
         * The threshold scales with the number of processors and the amount
         * of memory per zone. More memory means that we can defer updates for
         * longer, more processors could lead to more contention.
         * fls() is used to have a cheap way of logarithmic scaling.
         *
         * Some sample thresholds:
         *
         * Threshold    Processors      (fls)   Zonesize        fls(mem+1)
         * ------------------------------------------------------------------
         * 8            1               1       0.9-1 GB        4
         * 16           2               2       0.9-1 GB        4
         * 20           2               2       1-2 GB          5
         * 24           2               2       2-4 GB          6
         * 28           2               2       4-8 GB          7
         * 32           2               2       8-16 GB         8
         * 4            2               2       <128M           1
         * 30           4               3       2-4 GB          5
         * 48           4               3       8-16 GB         8
         * 32           8               4       1-2 GB          4
         * 32           8               4       0.9-1GB         4
         * 10           16              5       <128M           1
```

```
         * 40          16              5       900M           4
         * 70          64              7       2-4 GB         5
         * 84          64              7       4-8 GB         6
         * 108         512             9       4-8 GB         6
         * 125         1024            10      8-16 GB        8
         * 125         1024            10      16-32 GB       9
         */

        mem = zone_managed_pages(zone) >> (27 - PAGE_SHIFT);

        threshold = 2 * fls(num_online_cpus()) * (1 + fls(mem));

        /*
         * Maximum threshold is 125
         */
        threshold = min(125, threshold);

        return threshold;
}
```

Proportional to the size of the managed_pages in the request zone, the maximum number of threads to be used when under memory pressure is limited to a maximum of 125.

- Calculation Formula
  - threshold = 2 * (log2(number of online cpus) + 1) * (1 + log2(z->managed_pages / 128M) + 1)
    - = 2 * fls(number of online CPUs) + fls(z->managed_pages / 128M)
  - Threshold values are limited to a maximum of 125
  - e.g. 6 CPUs, zone is managed_pages=3.5G
    - threshold=2 x 3 x 5=30
  - e.g. 64 cpu, zone is managed_pages=3.5G
    - threshold=2 x 7 x 5=70

b) Calculating threads to be used for memory compression

The pressure thread value is the thread value used immediately after the kswapd is woken up due to insufficient memory, and the related function is as follows.

- kswapd_try_to_sleep() function

calculate_pressure_threshold()

mm/vmstat.c

```
int calculate_pressure_threshold(struct zone *zone)
{
        int threshold;
        int watermark_distance;

        /*
         * As vmstats are not up to date, there is drift between the estimated
         * and real values. For high thresholds and a high number of CPUs, it
         * is possible for the min watermark to be breached while the estimated
         * value looks fine. The pressure threshold is a reduced value such
```

```
         * that even the maximum amount of drift will not accidentally b
reach
         * the min watermark
         */
        watermark_distance = low_wmark_pages(zone) - min_wmark_pages(zon
e);
        threshold = max(1, (int)(watermark_distance / num_online_cpus
()));

        /*
         * Maximum threshold is 125
         */
        threshold = min(125, threshold);

        return threshold;
}
```

Limit the maximum number of threads to be used under memory pressure in proportion to the size of the managed_pages in the request zone, up to 125.

- In a system with a large number of CPUs, it is possible that there are a lot of PER-CPU counters that are maintained on each CPU that are not reflected in the zone counters. Therefore, in a low-memory situation, the following calculation formula is used to reduce the threaded value.
- Calculation Formula
  - threshold = interval between watermarks / number of CPUs online
  - Threshold values are limited to a maximum of 125
  - e.g. 6 CPUs, min=1975, low=2468, high=2962,
    - threshold=493 / 6 = 82
  - e.g. 64 CPUs, min=1975, low=2468, high=2962
    - threshold=493 / 64 = 7

Lowmem Reserve Page Output

The way to specify the desired zone when requesting a memory allocation is to use the following GFP mask. If the following flags are not used, the NORMAL zone is selected by default.

- __GFP_DMA
- __GFP_DMA32
- __GFP_HIGHMEM
- __GFP_MOVABLE

Kernel developers use the following GFP mask to select zones when allocating memory:

- GFP_DMA
  - __GFP_DMA Use the gfp flag and select the DMA zone.
- GFP_DMA32
  - __GFP_DMA32 Use the gfp flag and select the DMA32 zone.
- GFP_KERNEL
  - If you don't use the gfp flag associated with zone assignment, select the NORMAL zone.

- GFP_HIGHUSER
  - __GFP_HIGHMEM Use the gfp flag and select the HIGHMEM zone.
- GFP_HIGHUSER_MOVABLE
  - Use the __GFP_HIGHMEM and __GFP_HIGHUSER gfp flags, and select the MOVABLE zone.

If the requested zone fails to be allocated, it will be dropped back to the next zone via the zonelist, in the following order:

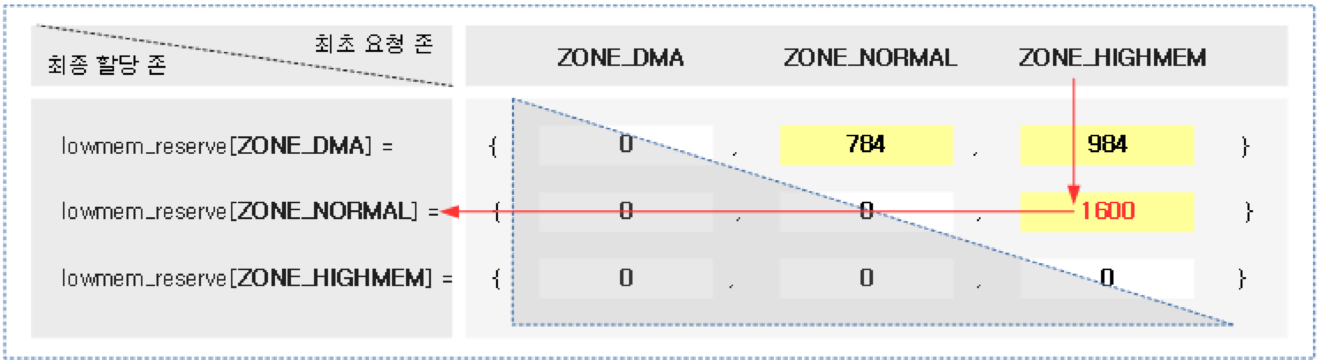
- MOVABLE -> HIGHMEM -> NORMAL -> DMA32 -> DMA

lowmem_reserve

The memory regions used in the normal and DMA/DMA32 zones, which are lowmem zones among other zones, are pre-mapped to the kernel and allow for fast kernel memory allocation. Therefore, it is not preferred to allocate lowmem areas when requesting allocation for highmem and movable zones used by user applications. If the parent zone inevitably falls back due to insufficient memory and needs to be allocated in a zone in the lowmem zone, the number of pages to limit the allocation of that zone is specified. These values are specified as a matrix for the initial request zone and the final target zone.

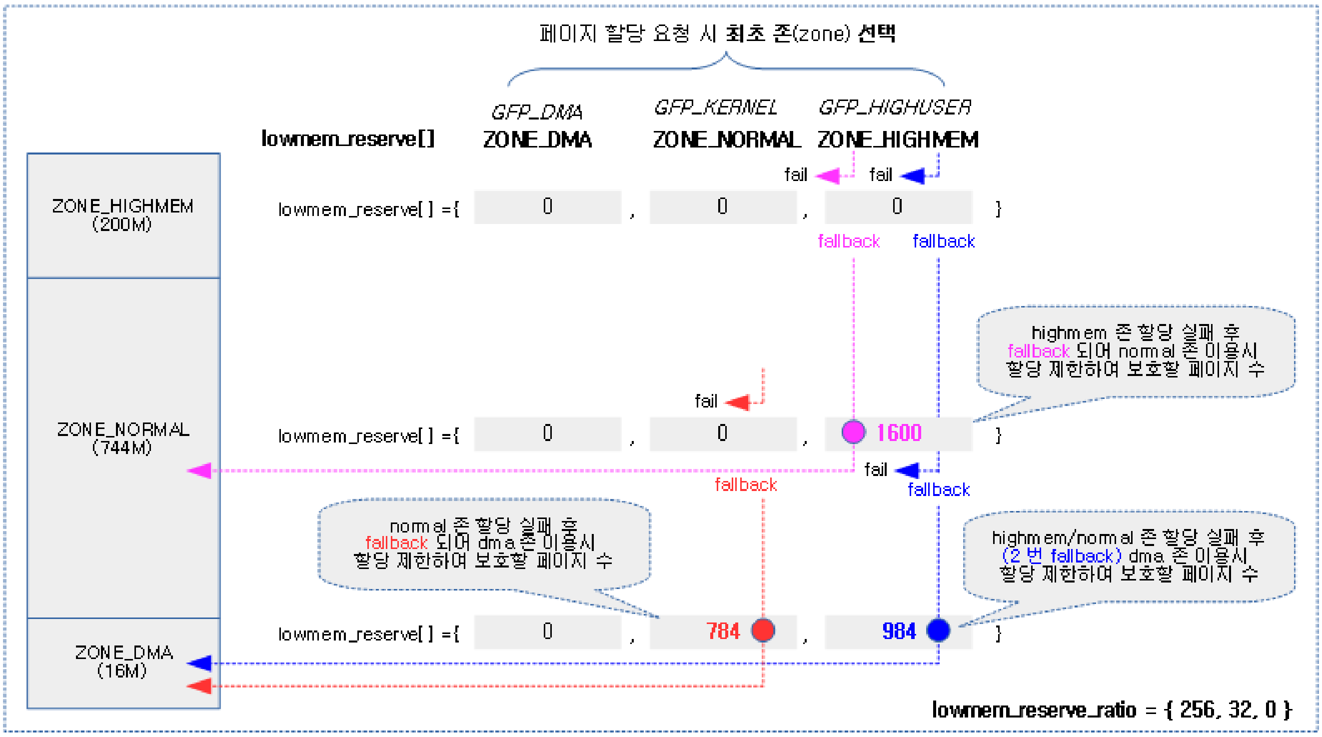
As shown in the following figure, refer to the lowmem_reserve values of each zone.

- It is assumed that an ARM1 system with 32G memory is running three zones at dma (3M), normal (16M), and high mem (784M) respectively.
- The values in parentheses {} are the order of zones operated by the system, and those values specify the number of pages allowed to be assigned.
- If you follow the red line below, you can see that when you receive a highmem request but it fallbacks and you head to the normal zone, you can protect the lowmem area by restricting the allocation by raising it by 1600 pages higher than the watermark threshold.



(http://jake.dothome.co.kr/wp-content/uploads/2016/06/lowmem_reserve-3.png)

As shown in the following figure, when the allocation fails in the zone initially requested when the page is allocated, it is fallback, and when it goes to the lowmem area, it shows the page allocation limit that is added more than the watermark criterion.



(http://jake.dothome.co.kr/wp-content/uploads/2016/06/lowmem_reserve-1c.png)

lowmem_reserve_ratio

It is a ratio for calculating the lowmem_reserve value for each zone using the managed_pages of each zone.

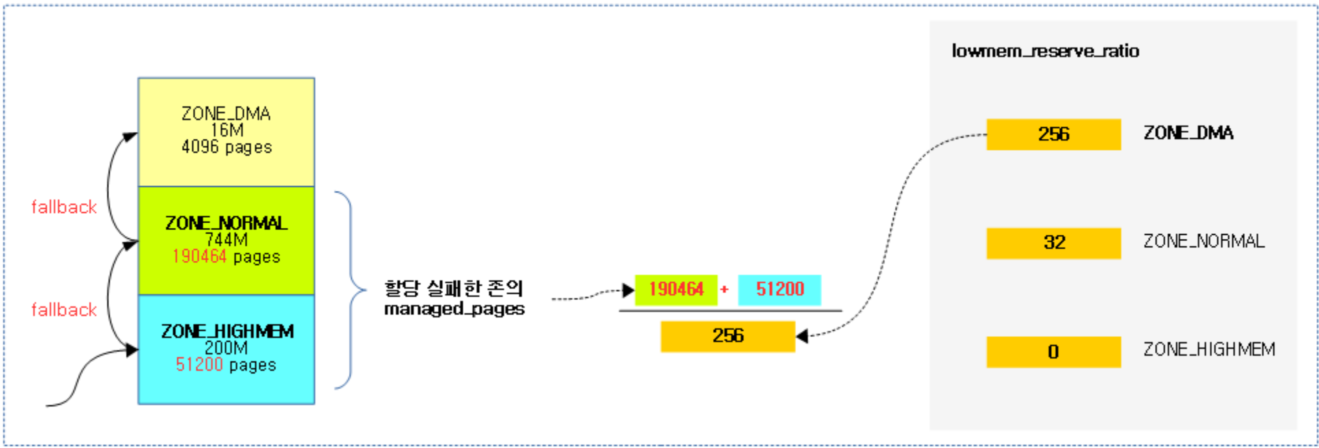
You can check the lowmem_reserve_ratio values as follows:

- The ratio is set in order from the lowest to the highest zone operated in the system.
- The example values below are in the order of DMA, Normal, and Highmem zones.

```
$ cat /proc/sys/vm/lowmem_reserve_ratio
256     32      0
```

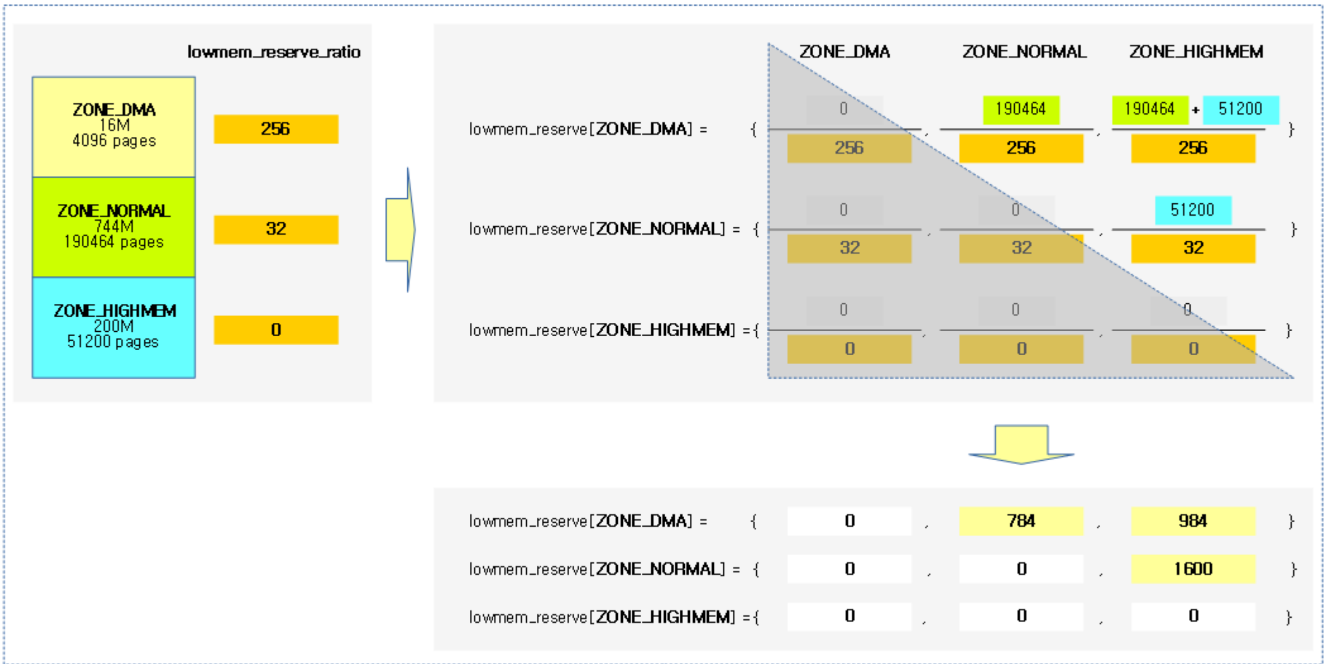
As shown in the following figure, the calculation of the number of pages allowed when fallback for the allocation request zone is as follows:

- It can be seen that the value of the numerator adds the number of managed_pages from the request zone to the allocation failure zone.
- It can be seen that it is fallback to the value of the denominator, specifying the lowmem_reserve_ratio of the final allocation zone.



(http://jake.dothome.co.kr/wp-content/uploads/2016/06/lowmem_reserve-4.png)

As shown in the following figure, the managed_pages size and lowmem_reserve_ratio of each zone are used to calculate the lowmem_reserve value of each zone.



(http://jake.dothome.co.kr/wp-content/uploads/2016/06/lowmem_reserve-2.png)

Next, let's look at the node-zone lowmem_reserve values on an x4 system with NUMA and four zones (dma, dma32, normal, movable).

- Take a look at the protection section below. When this feature was first developed, the variable was called protection, which was later changed to lowmem_reserve.

```
$ cat /proc/zoneinfo
Node 0, zone      DMA
  pages free     3915
        min      5
        low      6
        high     7
        scanned  0
        spanned  4095
        present  3992
        managed  3971
    nr_free_pages 3915
```

```
...
        protection: (0, 1675, 31880, 31880)
...
Node 0, zone    DMA32
  pages free     59594
        min      588
        low      735
        high     882
        scanned  0
        spanned  1044480
        present  491295
        managed  429342
    nr_free_pages 59594
...
        protection: (0, 0, 30204, 30204)
...
Node 0, zone   Normal
  pages free     902456
        min      10607
        low      13258
        high     15910
        scanned  0
        spanned  7864320
        present  7864320
        managed  7732469
    nr_free_pages 902456
...
        protection: (0, 0, 0, 0)
...
Node 1, zone   Normal
  pages free     1133093
        min      11326
        low      14157
        high     16989
        scanned  0
        spanned  8388608
        present  8388608
        managed  8256697
    nr_free_pages 1133093
...
        protection: (0, 0, 0, 0)
...
```

You can see that the lowmem_reserve value in a system with only one zone is 0 as follows.

- In the case of an ARM32 system that operates the dma64 and normal zones, it operates only one zone if all the memory is used with only the dma32 zone. In other words, there is no fallback.

```
$ cat /proc/zoneinfo
Node 0, zone    DMA32
  per-node stats
(...생략...)
  pages free     633680
        min      5632
        low      7040
        high     8448
        spanned  786432
        present  786432
        managed  765771
        protection: (0, 0, 0)     <-----
      nr_free_pages 633680
(...생략...)
Node 0, zone   Normal
  pages free     0
        min      0
        low      0
        high     0
        spanned  0
        present  0
        managed  0
        protection: (0, 0, 0)     <-----
(...생략...)
```

setup_per_zone_lowmem_reserve()

mm/page_alloc.c

```
/*
 * setup_per_zone_lowmem_reserve - called whenever
 *      sysctl_lower_zone_reserve_ratio changes.  Ensures that each zone
 *      has a correct pages reserved value, so an adequate number of
 *      pages are left in the zone after a successful __alloc_pages().
 */
```

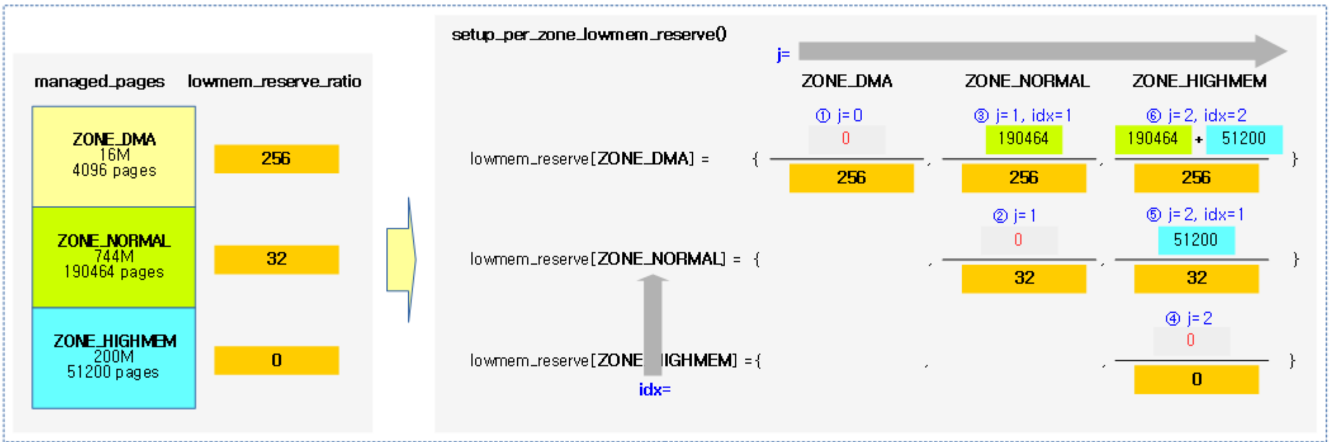
```
static void setup_per_zone_lowmem_reserve(void)
{
        struct pglist_data *pgdat;
        enum zone_type j, idx;

        for_each_online_pgdat(pgdat) {
                for (j = 0; j < MAX_NR_ZONES; j++) {
                        struct zone *zone = pgdat->node_zones + j;
                        unsigned long managed_pages = zone_managed_pages(zone);

                        zone->lowmem_reserve[j] = 0;

                        idx = j;
                        while (idx) {
                                struct zone *lower_zone;

                                idx--;
                                lower_zone = pgdat->node_zones + idx;

                                if (sysctl_lowmem_reserve_ratio[idx] < 1) {
                                        sysctl_lowmem_reserve_ratio[idx] = 0;
```

```
22                                      lower_zone->lowmem_reserve[j] = 0;
23                              } else {
24                                      lower_zone->lowmem_reserve[j] =
25                                              managed_pages / sysctl_lowmem_reserve_ratio[idx];
26                              }
27                              managed_pages += zone_managed_pages(lower_zone);
28                      }
29              }
30      }
31
32      /* update totalreserve_pages */
33      calculate_totalreserve_pages();
34 }
```

The managed_pages value of the zone and the ratio of the lowmem_reserve_ratio are used to calculate the lowmem_reserve[] page values for each zone.

- In code lines 6~11, traverse the total number of online nodes, internally traverse the number of available zones in that node, and reset the lowmem_reserve[] value to 0.
- In code lines 13~30, go around the loop from the zone designated as the loop counter except for the first zone, and add the managed_pages except the first zone (idx=0) to specify the lowmem_reserve_ratio ratio to the lowmem_reserve.
- Since the lowmem_reserve value has changed in line 33 of code, the number of TotalReserve pages is also updated.

The following figure shows the process of calculating the lowmem_reserve[] for each zone.



(http://jake.dothome.co.kr/wp-content/uploads/2016/06/setup_per_zone_lowmem_reserve-2.png)

TotalReserve Page Output

calculate_totalreserve_pages()

mm/page_alloc.c

```
/*
 * calculate_totalreserve_pages - called when sysctl_lower_zone_reserve_ratio
 *      or min_free_kbytes changes.
```

```
 */
```

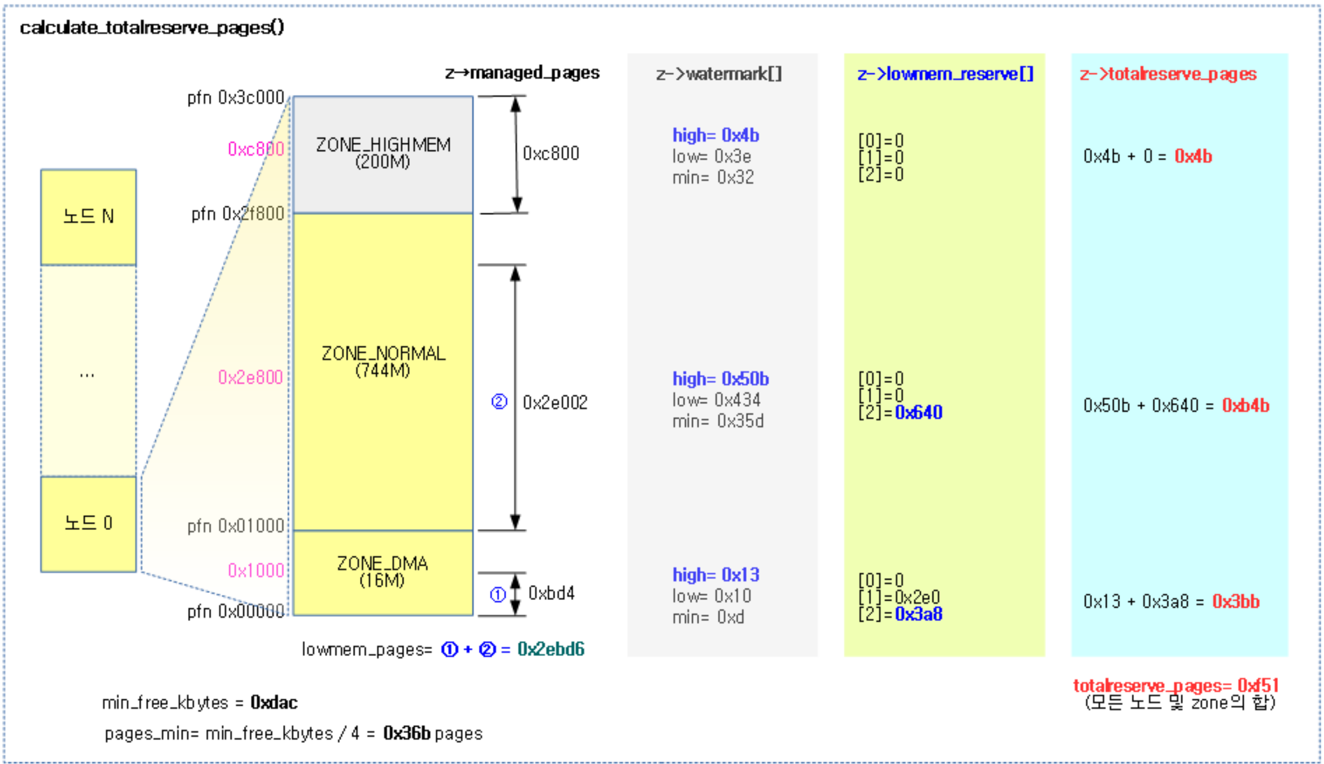
```
01 static void calculate_totalreserve_pages(void)
02 {
03         struct pglist_data *pgdat;
04         unsigned long reserve_pages = 0;
05         enum zone_type i, j;
06 
07         for_each_online_pgdat(pgdat) {
08 
09                 pgdat->totalreserve_pages = 0;
10 
11                 for (i = 0; i < MAX_NR_ZONES; i++) {
12                         struct zone *zone = pgdat->node_zones + i;
13                         long max = 0;
14                         unsigned long managed_pages = zone_managed_pages(zone);
15 
16                         /* Find valid and maximum lowmem_reserve in the zone */
17                         for (j = i; j < MAX_NR_ZONES; j++) {
18                                 if (zone->lowmem_reserve[j] > max)
19                                         max = zone->lowmem_reserve[j];
20                         }
21 
22                         /* we treat the high watermark as reserved pages. */
23                         max += high_wmark_pages(zone);
24 
25                         if (max > managed_pages)
26                                 max = managed_pages;
27 
28                         pgdat->totalreserve_pages += max;
29 
30                         reserve_pages += max;
31                 }
32         }
33         totalreserve_pages = reserve_pages;
34 }
```

Calculate the totalreserve_pages value for each node and the global totalreserve_pages value.

- In line 7~9 of the code, it traverses all nodes and initializes them to 0 first to calculate the totalreserve_pages.
- In lines 11~14 of code, traverse all zones and find out the managed page of that zone.
- From code lines 17~28, find out the maximum value of the lowmem_reserve set from the traversing zone to the last zone, and then add a high watermark to add it to the totalreserve_pages. Limit the value to be added so that it does not exceed the managed_pages of the zone in the circuit.
- In line 30~33 of the code, assign the values added by each node and zone to the global totalreserve_pages.

The following figure shows the process of calculating the following values by adding each node and zone to calculate the totalreserve_pages value.

- Maximum lowmem_reserve above each zone
- High watermark for each zone



(http://jake.dothome.co.kr/wp-content/uploads/2016/06/calculate_totalreserve_pages-1b.png)

---

inactive_ratio Output

The following steps removed the inactive_ratio from kernel v4.7-rc1 on bootup. The proportion of inactive lists is now deprecated because it is automatically compared to the entire memory in the inactive_list_is_low() function.

- 참고: mm: vmscan: reduce size of inactive file list (https://github.com/torvalds/linux/commit/59dc76b0d4dfdd7dc46a1010e4afb44f60f3e97f#diff-d1793bdc2ce9e21814061bb882d0c3ac)

setup_per_zone_inactive_ratio()

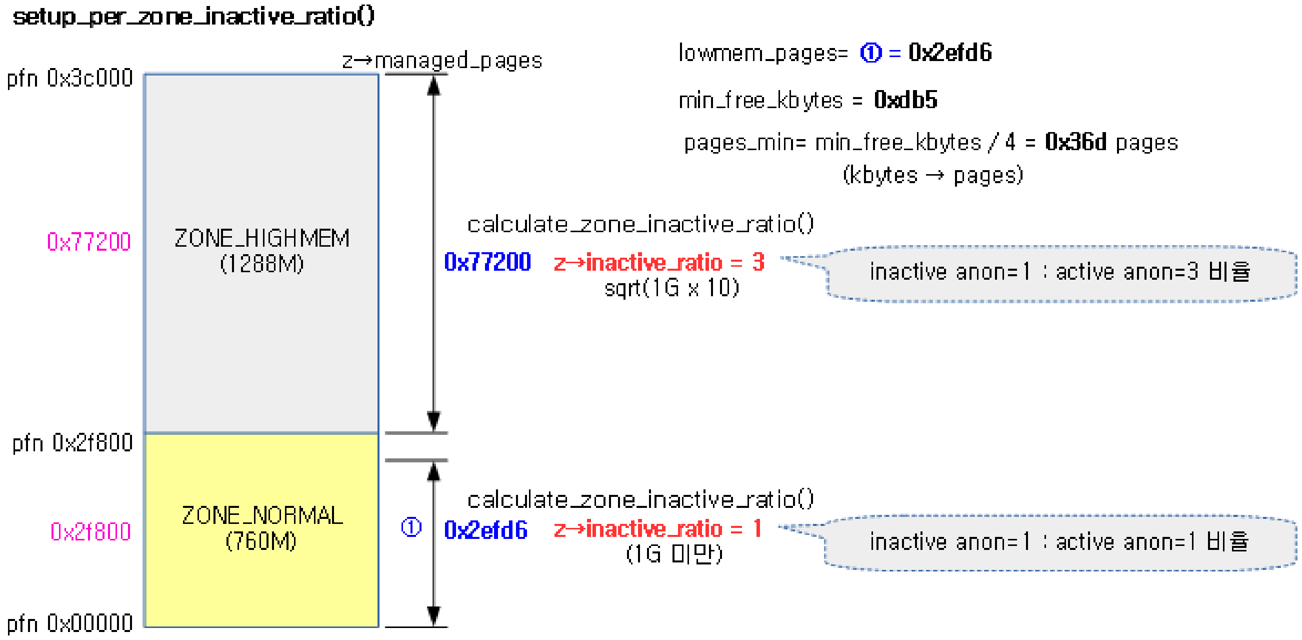
mm/page_alloc.c

```
static void __meminit setup_per_zone_inactive_ratio(void)
{
        struct zone *zone;

        for_each_zone(zone)
                calculate_zone_inactive_ratio(zone);
}
```

Calculate the inactive anon rate of all zones.

- If the zone->managed_pages is less than 256k pages (1GB), the zone->inactive_ratio will be 1 and the ratio of active and inactive anon will be set to 1:1. If the managed_pages exceeds 256k pages (1GB), the inactive_ratio value will be greater than 3 and the percentage of inactive anon will be reduced to less than 1/3.

The following figure shows the process of calculating inactive_ratio values based on the managed_pages size of each zone.



(http://jake.dothome.co.kr/wp-content/uploads/2016/06/setup_per_zone_inactive_ratio-1.png)

calculate_zone_inactive_ratio()

mm/page_alloc.c

```
/*
 * The inactive anon list should be small enough that the VM never has t
o
 * do too much work, but large enough that each inactive page has a chan
ce
 * to be referenced again before it is swapped out.
 *
 * The inactive_anon ratio is the target ratio of ACTIVE_ANON to
 * INACTIVE_ANON pages on this zone's LRU, maintained by the
 * pageout code. A zone->inactive_ratio of 3 means 3:1 or 25% of
 * the anonymous pages are kept on the inactive list.
 *
 * total     target    max
 * memory    ratio     inactive anon
 * -------------------------------------
 *   10MB       1         5MB
 *  100MB       1        50MB
 *    1GB       3       250MB
 *   10GB      10       0.9GB
 *  100GB      31         3GB
 *    1TB     101        10GB
 *   10TB     320        32GB
 */
static void __meminit calculate_zone_inactive_ratio(struct zone *zone)
{
        unsigned int gb, ratio;

        /* Zone size in gigabytes */
        gb = zone->managed_pages >> (30 - PAGE_SHIFT);
        if (gb)
                ratio = int_sqrt(10 * gb);
        else
```

```
                ratio = 1;

        zone->inactive_ratio = ratio;
}
```

Calculate the inactive anon rate of the specified zone.

- If the managed page is less than 1G, the ratio will be 1.
  - inactive anon=1 : active anon=1
- If the managed page is more than 1G, set the ratio by using * 10 and then the square root (root). (inactive = 1 / ratio)
  - 예) managed pages=1G
    - inactive_ratio=3
      - inactive anon=1 : active anon=3
        - inactive anon=250M : active anon=750M

consultation

- Zoned Allocator -1- (Physics Page Assignment - Fastpath) (http://jake.dothome.co.kr/zonned-allocator-alloc-pages-fastpath) | Qc
- Zoned Allocator -2- (Physics Page Assignment - Slowpath) (http://jake.dothome.co.kr/zonned-allocator-alloc-pages-slowpath) | Qc
- Zoned Allocator -3- (Buddy Page Allocation) (http://jake.dothome.co.kr/buddy-alloc) | Qc
- Zoned Allocator -4- (Buddy Page Terminated) (http://jake.dothome.co.kr/buddy-free/) | Qc
- Zoned Allocator -5- (Per-CPU Page Frame Cache) (http://jake.dothome.co.kr/per-cpu-page-frame-cache) | 문c
- Zoned Allocator -6- (Watermark) (http://jake.dothome.co.kr/zonned-allocator-watermark) | Sentence C – Current post
- Zoned Allocator -7- (Direct Compact) (http://jake.dothome.co.kr/zonned-allocator-compaction) | 문c
- Zoned Allocator -8- (Direct Compact-Isolation) (http://jake.dothome.co.kr/zonned-allocator-isolation) | 문c
- Zoned Allocator -9- (Direct Compact-Migration) (http://jake.dothome.co.kr/zonned-allocator-migration) | 문c
- Zoned Allocator -10- (LRU & pagevec) (http://jake.dothome.co.kr/lru-lists-pagevecs) | 문c
- Zoned Allocator -11- (Direct Reclaim) (http://jake.dothome.co.kr/zonned-allocator-reclaim) | 문c
- Zoned Allocator -12- (Direct Reclaim-Shrink-1) (http://jake.dothome.co.kr/zonned-allocator-shrink-1) | 문c
- Zoned Allocator -13- (Direct Reclaim-Shrink-2) (http://jake.dothome.co.kr/zonned-allocator-shrink-2) | 문c
- Zoned Allocator -14- (Kswapd) (http://jake.dothome.co.kr/zonned-allocator-kswapd) | 문c

11 thoughts to “Zoned Allocator -6- (Watermark)”

AUSTIN KIM (HTTP://ROUSALOME.EGLOOS.COM/)
2020-01-13 14:18 (http://jake.dothome.co.kr/zonned-allocator-watermark/#comment-228690)

This is Moon Young-il and Kim Dong-hyun.

I'm studying and learning a lot from the blog. Thank you as always.
However, I saw a typo in line 3 of the WATERMARK_MIN entry. I would like to contribute to the 'Munc Blog'.
'Page recall price -> page reclaim'

RESPONSE (/ZONNED-ALLOCATOR-WATERMARK/?REPLYTOCOM=228690#RESPOND)

MOON YOUNG-IL (HTTP://JAKE.DOTHOME.CO.KR)
2020-01-15 07:53 (http://jake.dothome.co.kr/zonned-allocator-watermark/#comment-228903)

Donghyun, how did you spend the end of the year?

In my first job in the 1990s, I wondered, 'Will the year 2000 come?'
But every day, I see the number 2000, which I thought would never come, let alone 2020.
It's a big new year with a two-digit change, so I wish Donghyun a Happy New Year as well.
I'm sure you're busy with the publication of your kernel book, and I hope your life will be richer this year.

We hope to see you often. ^^

RESPONSE (/ZONNED-ALLOCATOR-WATERMARK/?REPLYTOCOM=228903#RESPOND)

AUSTIN KIM (HTTP://ROUSALOME.EGLOOS.COM)
2020-01-16 21:44 (http://jake.dothome.co.kr/zonned-allocator-watermark/#comment-228987)

Thank you for your support.
It's late, but I wish you a happy new year and I hope to see you again this year with a different opportunity.

Like the 'Munc Blog', I will study kernel diligently this year.

RESPONSE (/ZONNED-ALLOCATOR-WATERMARK/?REPLYTOCOM=228987#RESPOND)

PETER

2020-12-13 13:48 (http://jake.dothome.co.kr/zonned-allocator-watermark/#comment-294600)

I have a question in the __zone_watermark_ok() function.

"Reduce the number of pages requested to be allocated free pages on line 10 by -1."
If you count the number of pages with the requested order, subtract 1 and put it in the free_pages,
why do you do -1 here?

응답 (/ZONNED-ALLOCATOR-WATERMARK/?REPLYTOCOM=294600#RESPOND)

문영일 (HTTP://JAKE.DOTHOME.CO.KR)
2020-12-13 14:54 (http://jake.dothome.co.kr/zonned-allocator-watermark/#comment-294609)

안녕하세요?

저도 예전에 한 번 왜 -1을 할 까 생각을 한적이 있었습니다만
개발자의 정확한 의미는 모르고 지나간 적이 있었습니다. 다만 아래 코드와 같이 코드를 작성해도 되는데

```
free_pages -= (1 < < order); ... if ((free_pages + 1) <= min + z->lowmem_reserve[classzone_idx])
```

커널 코드는 다음과 같이 미리 1을 덜 빼두고 비교 판단 코드를 작성하기 위함이라 판단됩니다.

감사합니다.

```
free_pages -= (1 < < order) - 1; ... if (free_pages <= min + z->lowmem_reserve[classzone_idx])
```

응답 (/ZONNED-ALLOCATOR-WATERMARK/?REPLYTOCOM=294609#RESPOND)

PETER
2020-12-13 19:44 (http://jake.dothome.co.kr/zonned-allocator-watermark/#comment-294662)

생각해 보니 아래와 같이 등호(=)를 빼는 방법도 가능해 보입니다.

free_pages -= (1 < < order); ... if ((free_pages) lowmem_reserve[classzone_idx])

말씀하신 위치의 비교 조건에만 관련된 구현의 차이 일뿐,
알고리즘상의 큰 차이는 없어 보입니다.

감사합니다.

응답 (/ZONNED-ALLOCATOR-WATERMARK/?REPLYTOCOM=294662#RESPOND)

PETER
2020-12-13 19:46 (http://jake.dothome.co.kr/zonned-allocator-watermark/#comment-294663)

if (free_pages lowmem_reserve[classzone_idx])
입니다

응답 (/ZONNED-ALLOCATOR-WATERMARK/?REPLYTOCOM=294663#RESPOND)

문영일 (HTTP://JAKE.DOTHOME.CO.KR)
2020-12-14 06:36 (http://jake.dothome.co.kr/zonned-allocator-watermark/#comment-294778)

네. 부등호(=)를 빼도 될 듯 합니다. ^^

응답 (/ZONNED-ALLOCATOR-WATERMARK/?REPLYTOCOM=294778#RESPOND)

메모리바보
2021-01-22 22:35 (http://jake.dothome.co.kr/zonned-allocator-watermark/#comment-303303)

안녕하세요,

좋은 정보 감사합니다.

본문에 나오는 lowmem은 zone_highmem을 제외한 영역인가요? 그렇다면 lowmem을 기준으로 watermark를 산정하는 이유는 무엇인가요?

응답 (/ZONNED-ALLOCATOR-WATERMARK/?REPLYTOCOM=303303#RESPOND)

문영일 (HTTP://JAKE.DOTHOME.CO.KR)
2021-01-23 14:09 (http://jake.dothome.co.kr/zonned-allocator-watermark/#comment-303487)

안녕하세요?

커널에서 사용되는 lowmem은 다음 두 가지 의미로 사용됩니다.
1) 메모리 존을 뜻할 때에는 highmem 존의 반대 의미로 사용되고, lowmem에는 normal, dma32, dma 존이 포함됩니다. (64bit의 경우 highmem이 존재하지 않습니다)
2) 메모리 부족 경계를 의미로 사용되는 lowmem도 있습니다. 따라서 이 기준에 워터마크를 사용합니다.

감사합니다.

응답 (/ZONNED-ALLOCATOR-WATERMARK/?REPLYTOCOM=303487#RESPOND)

메모리바보
2021-01-23 19:24 (http://jake.dothome.co.kr/zonned-allocator-watermark/#comment-303536)

아 두가지가 다른 의미를 가지는 군요.
좋은 정보 감사합니다.!

응답 (/ZONNED-ALLOCATOR-WATERMARK/?REPLYTOCOM=303536#RESPOND)

댓글 남기기

이메일은 공개되지 않습니다. 필수 입력창은 * 로 표시되어 있습니다

댓글

이름 *

이메일 *

웹사이트

댓글 작성

< Zoned Allocator -2- (물리 페이지 할당-Slowpath) (http://jake.dothome.co.kr/zonned-allocator-alloc-pages-slowpath/)

Zoned Allocator -7- (Direct Compact) > (http://jake.dothome.co.kr/zonned-allocator-compaction/)