VMPressure

2019-09-17 (http://jake.dothome.co.kr/vmpressure/) Moon Young-il (http://jake.dothome.co.kr/author/admin/) Linux Kernel (http://jake.dothome.co.kr/category/linux/)

<kernel v5.0>

VMPressure

The memory pressure rate is calculated by analyzing the ratio of pages scanned and recalled through the Memory Control Ggroup, and the vmpressure listeners registered in memcg can be notified at three event levels for each thread. VMPurstionListeners can use EventFD to receive these events.

Event Levels

- low
  - The memory pressure specified by memcg is low.
- medium
  - There is a lot of memory pressure specified by memcg.
- critical
  - The memory pressure specified by memcg is severe, and the OOM killer will be running soon.

vmpressure_win

mm/vmpressure.c

```
01 /*
02  * The window size (vmpressure_win) is the number of scanned pages befor
   e
03  * we try to analyze scanned/reclaimed ratio. So the window is used as a
04  * rate-limit tunable for the "low" level notification, and also for
05  * averaging the ratio for medium/critical levels. Using small window
06  * sizes can cause lot of false positives, but too big window size will
07  * delay the notifications.
08  *
09  * As the vmscan reclaimer logic works with chunks which are multiple of
10  * SWAP_CLUSTER_MAX, it makes sense to use it for the window size as wel
   l.
11  *
12  * TODO: Make the window size depend on machine size, as we do for vmsta
   t
13  * thresholds. Currently we set it to 512 pages (2MB for 4KB pages).
14  */
```

```
1 static const unsigned long vmpressure_win = SWAP_CLUSTER_MAX * 16;
```

- SWAP_CLUSTER_MAX(32) * 16 = 512 pages.

- This window size is the number of pages scanned to use before attempting to analyze the scanned/reclaim ratio.
- It is used for low-level notifications and for the average ratio of medium/critical levels.

vmpressure_level_med & vmpressure_level_critical

mm/vmpressure.c

```
/*
 * These thresholds are used when we account memory pressure through
 * scanned/reclaimed ratio. The current values were chosen empirically.
In
 * essence, they are percents: the higher the value, the more number
 * unsuccessful reclaims there were.
 */
```

```
static const unsigned int vmpressure_level_med = 60;
static const unsigned int vmpressure_level_critical = 95;
```

- vmpressure_level_med
  - The scanned / reclaimed ratio of the thread value at the medium level used when weighing memory pressure
- vmpressure_level_critical
  - The scanned/reclaimed ratio, the threaded value of the critical level used when weighing memory pressure

vmpressure_prio()

mm/vmpressure.c

```
/**
 * vmpressure_prio() - Account memory pressure through reclaimer priorit
y level
 * @gfp:        reclaimer's gfp mask
 * @memcg:      cgroup memory controller handle
 * @prio:       reclaimer's priority
 *
 * This function should be called from the reclaim path every time when
 * the vmscan's reclaiming priority (scanning depth) changes.
 *
 * This function does not return any value.
 */
```

```
void vmpressure_prio(gfp_t gfp, struct mem_cgroup *memcg, int prio)
{
        /*
         * We only use prio for accounting critical level. For more info
         * see comment for vmpressure_level_critical_prio variable abov
e.
         */
        if (prio > vmpressure_level_critical_prio)
                return;

        /*
         * OK, the prio is below the threshold, updating vmpressure
         * information before shrinker dives into long shrinking of long
         * range vmscan. Passing scanned = vmpressure_win, reclaimed = 0
         * to the vmpressure() basically means that we signal 'critical'
         * level.
```

```
16          */
17         vmpressure(gfp, memcg, true, vmpressure_win, 0);
18 }
```

If the priority is increased and the scan depth increases, the vmpressure information is updated.

- In line 7~8 of the code, the request priority is lower than vmpressure_level_critical_prio(3) and exits the function.
  - The lower the PRIO, the higher the priority.
- If PRIO drops below the thread in line 17 of the code, i.e. if it has increased priority, the shrinker will update the vmpressure information before scanning for a long time.

vmpressure()

mm/vmpressure.c

```
01 /**
02  * vmpressure() - Account memory pressure through scanned/reclaimed rati
   o
03  * @gfp:        reclaimer's gfp mask
04  * @memcg:      cgroup memory controller handle
05  * @tree:       legacy subtree mode
06  * @scanned:    number of pages scanned
07  * @reclaimed:  number of pages reclaimed
08  *
09  * This function should be called from the vmscan reclaim path to accoun
   t
10  * "instantaneous" memory pressure (scanned/reclaimed ratio). The raw
11  * pressure index is then further refined and averaged over time.
12  *
13  * If @tree is set, vmpressure is in traditional userspace reporting
14  * mode: @memcg is considered the pressure root and userspace is
15  * notified of the entire subtree's reclaim efficiency.
16  *
17  * If @tree is not set, reclaim efficiency is recorded for @memcg, and
18  * only in-kernel users are notified.
19  *
20  * This function does not return any value.
21  */
```

```
01 void vmpressure(gfp_t gfp, struct mem_cgroup *memcg, bool tree,
02                 unsigned long scanned, unsigned long reclaimed)
03 {
04         struct vmpressure *vmpr = memcg_to_vmpressure(memcg);
05
06         /*
07          * Here we only want to account pressure that userland is able t
   o
08          * help us with. For example, suppose that DMA zone is under
09          * pressure; if we notify userland about that kind of pressure,
10          * then it will be mostly a waste as it will trigger unnecessary
11          * freeing of memory by userland (since userland is more likely
   to
12          * have HIGHMEM/MOVABLE pages instead of the DMA fallback). That
13          * is why we include only movable, highmem and FS/IO pages.
14          * Indirect reclaim (kswapd) sets sc->gfp_mask to GFP_KERNEL, so
15          * we account it too.
16          */
17         if (!(gfp & (__GFP_HIGHMEM | __GFP_MOVABLE | __GFP_IO | __GFP_F
   S)))
18                 return;
19
20         /*
```

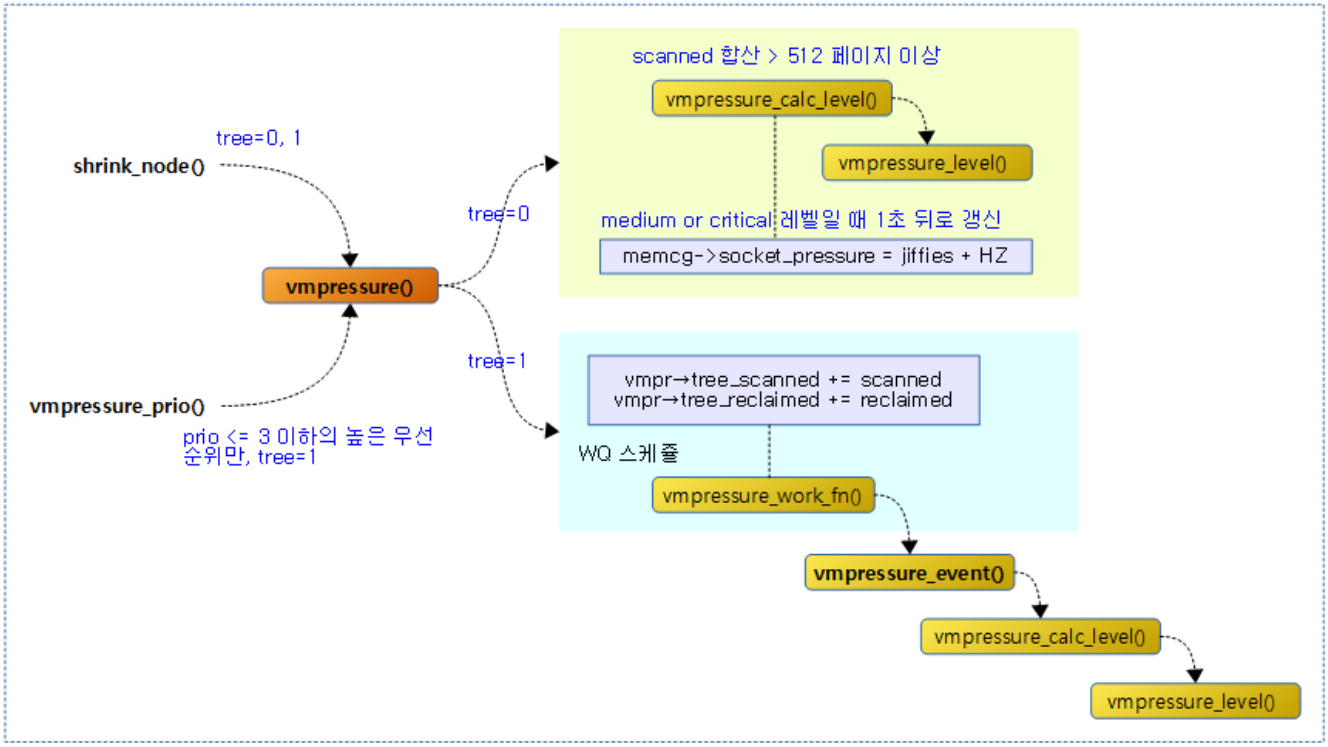
```
21          * If we got here with no pages scanned, then that is an indicat
    or
22          * that reclaimer was unable to find any shrinkable LRUs at the
23          * current scanning depth. But it does not mean that we should
24          * report the critical pressure, yet. If the scanning priority
25          * (scanning depth) goes too high (deep), we will be notified
26          * through vmpressure_prio(). But so far, keep calm.
27          */
28         if (!scanned)
29                 return;
30
31         if (tree) {
32                 spin_lock(&vmpr->sr_lock);
33                 scanned = vmpr->tree_scanned += scanned;
34                 vmpr->tree_reclaimed += reclaimed;
35                 spin_unlock(&vmpr->sr_lock);
36
37                 if (scanned < vmpressure_win)
38                         return;
39                 schedule_work(&vmpr->work);
40         } else {
41                 enum vmpressure_levels level;
42
43                 /* For now, no users for root-level efficiency */
44                 if (!memcg || memcg == root_mem_cgroup)
45                         return;
46
47                 spin_lock(&vmpr->sr_lock);
48                 scanned = vmpr->scanned += scanned;
49                 reclaimed = vmpr->reclaimed += reclaimed;
50                 if (scanned < vmpressure_win) {
51                         spin_unlock(&vmpr->sr_lock);
52                         return;
53                 }
54                 vmpr->scanned = vmpr->reclaimed = 0;
55                 spin_unlock(&vmpr->sr_lock);
56
57                 level = vmpressure_calc_level(scanned, reclaimed);
58
59                 if (level > VMPRESSURE_LOW) {
60                         /*
61                          * Let the socket buffer allocator know that
62                          * we are having trouble reclaiming LRU pages.
63                          *
64                          * For hysteresis keep the pressure state
65                          * asserted for a second in which subsequent
66                          * pressure events can occur.
67                          */
68                         memcg->socket_pressure = jiffies + HZ;
69                 }
70         }
71 }
```

Quantify memory pressure with scaned and reclaimed ratios.

- Returns the vmpressure information of the memcg requested in line 4.
- If there is no highmem, movable, FS, or IO flag request in code lines 17~18, no pressure metering is performed.
- In line 28~29 of the code, if the argument scanned is 0, the function is aborted.
- In code lines 31~39, measure the presssure of the existing tree method. Increase the tree_scanned and tree_reclaimed by that amount respectively and run the tasks registered with vmpr->work. If vmpr->scanned is less than vmpressure_win, the function is aborted.
    - vmpressure_work_fn()

- If the @tree is 40 in line 45~0 of the code, the recall efficiency for the @memcg is recorded in order to notify the user inside the kernel. If memcg is not specified, the function aborts.
- In lines 47~55 of the code, increment each of the scanned and reclaimed by that amount, and abort the function if the scanned is less than vmpressure_win. If not aborted, VMPR's Canned and Reclaimed resets to zero.
- If the vmpressure level calculated in code lines 57~69 exceeds VMPRESSURE_LOW, set the socket_pressure of the memcg to a tick value that is 1 second after the current time.
    - Use this value in the mem_cgroup_under_socket_pressure() function.
    - 참고: mm: memcontrol: hook up vmpressure to socket pressure (https://github.com/torvalds/linux/commit/8e8ae645249b85c8ed6c178557f8db8613a6bcc7)

The following figure shows how the vmpressure() function is processed.



(http://jake.dothome.co.kr/wp-content/uploads/2019/09/vmpressure-1.png)

Notification of events according to vmpressure in the work queue

vmpressure_work_fn()

mm/vmpressure.c

```
01 static void vmpressure_work_fn(struct work_struct *work)
02 {
03         struct vmpressure *vmpr = work_to_vmpressure(work);
04         unsigned long scanned;
05         unsigned long reclaimed;
06         enum vmpressure_levels level;
07         bool ancestor = false;
08         bool signalled = false;
09
```

```
10          spin_lock(&vmpr->sr_lock);
11          /*
12           * Several contexts might be calling vmpressure(), so it is
13           * possible that the work was rescheduled again before the old
14           * work context cleared the counters. In that case we will run
15           * just after the old work returns, but then scanned might be ze
    ro
16           * here. No need for any locks here since we don't care if
17           * vmpr->reclaimed is in sync.
18           */
19          scanned = vmpr->tree_scanned;
20          if (!scanned) {
21                  spin_unlock(&vmpr->sr_lock);
22                  return;
23          }
24
25          reclaimed = vmpr->tree_reclaimed;
26          vmpr->tree_scanned = 0;
27          vmpr->tree_reclaimed = 0;
28          spin_unlock(&vmpr->sr_lock);
29
30          level = vmpressure_calc_level(scanned, reclaimed);
31
32          do {
33                  if (vmpressure_event(vmpr, level, ancestor, signalled))
34                          signalled = true;
35                  ancestor = true;
36          } while ((vmpr = vmpressure_parent(vmpr)));
37  }
```

Calculate the memory pressure level and send an event to the vmpressure listener that satisfies the level and mode conditions.

- Get the tree_scanned and tree_reclaimed values from lines 10~28 of the code and reset them.
- In line 30 of code, calculate the level with the scanned value and the reclaimed value.
- In code lines 32~36, the vmpressure value of the memcg configured with the high raki is traversed to the top root, and the event is notified to the vmpressure listener that satisfies the condition.

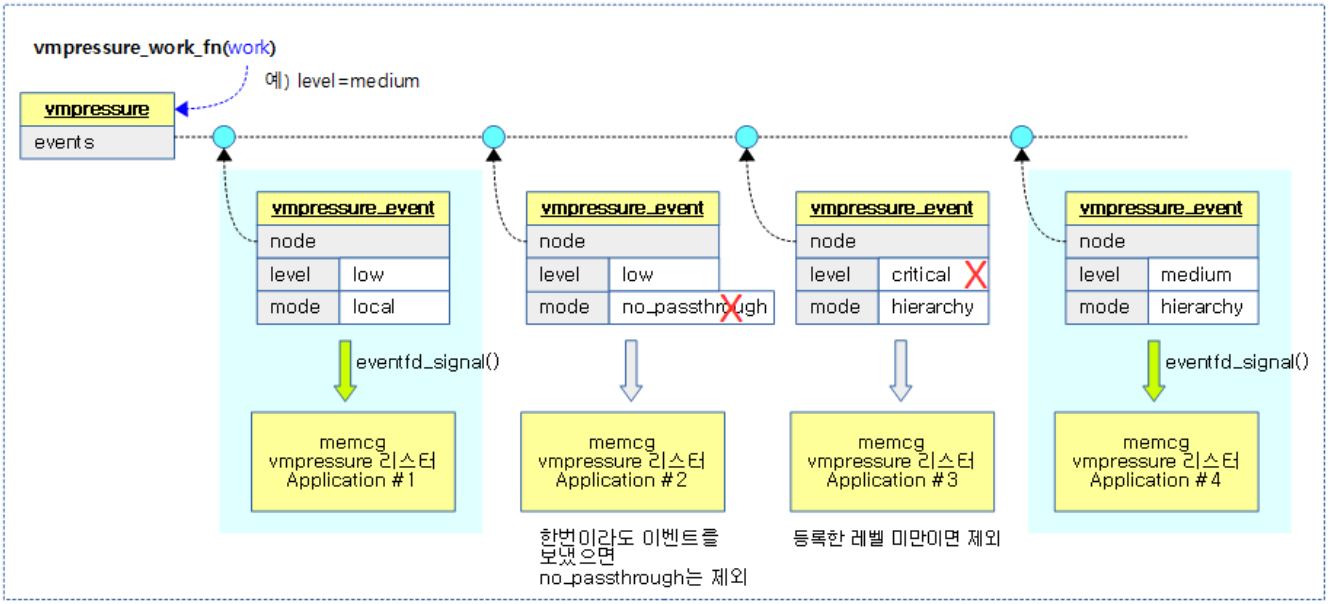
memcg_to_vmpressure()

mm/memcontrol.c

```
1   /* Some nice accessors for the vmpressure. */
2   struct vmpressure *memcg_to_vmpressure(struct mem_cgroup *memcg)
3   {
4           if (!memcg)
5                   memcg = root_mem_cgroup;
6           return &memcg->vmpressure;
7   }
```

Returns the vmpressure information of the requested memcg. If no memcg is specified, root returns the vmpressure of memcg.

The following illustration shows the process of sending an event to registered VMpressure listeners who meet the criteria.



(http://jake.dothome.co.kr/wp-content/uploads/2019/09/vmpressure_work_fn-1.png)

VMPurtion Event Notification

vmpressure_event()

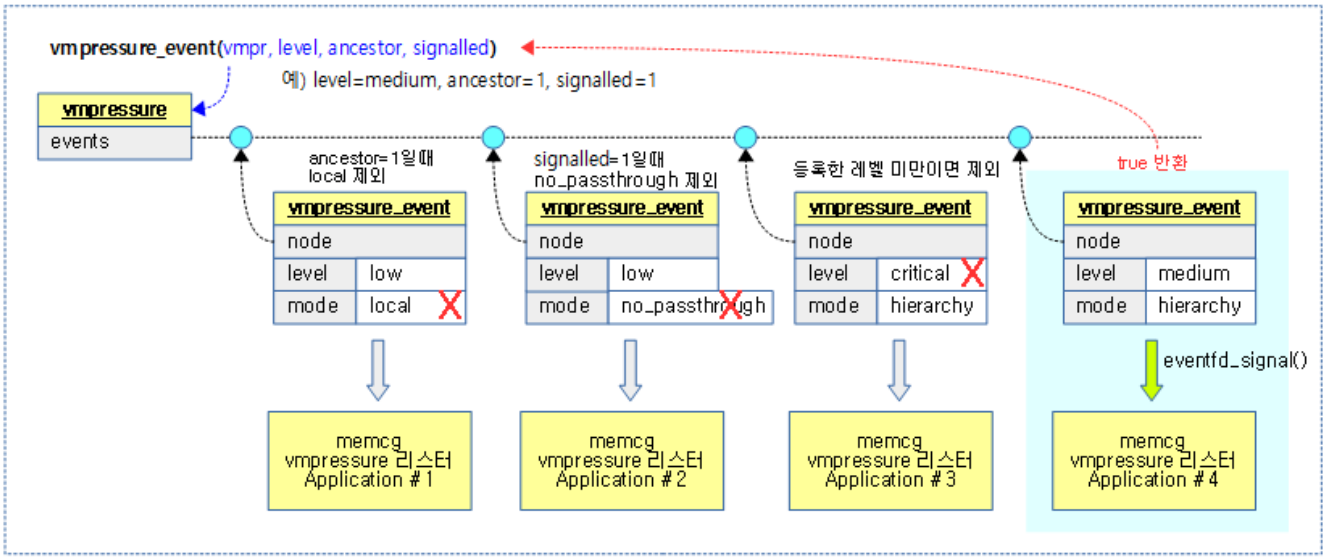
mm/vmpressure.c

```
01 static bool vmpressure_event(struct vmpressure *vmpr,
02                               const enum vmpressure_levels level,
03                               bool ancestor, bool signalled)
04 {
05         struct vmpressure_event *ev;
06         bool ret = false;
07
08         mutex_lock(&vmpr->events_lock);
09         list_for_each_entry(ev, &vmpr->events, node) {
10                 if (ancestor && ev->mode == VMPRESSURE_LOCAL)
11                         continue;
12                 if (signalled && ev->mode == VMPRESSURE_NO_PASSTHROUGH)
13                         continue;
14                 if (level < ev->level)
15                         continue;
16                 eventfd_signal(ev->efd, 1);
17                 ret = true;
18         }
19         mutex_unlock(&vmpr->events_lock);
20
21         return ret;
22 }
```

For events registered in VMpressure, the EventFD signal is notified to the VMpressure Lister Application that has registered less than @level the request.

- The following cases are not subject to notification:
  - When @ancestor=1, local mode is excluded.
  - When @signalled=1, no_passthrough mode is excluded.
  - Except for those who have registered at a level greater than @level.

The following figure shows the conditions under which the VMpressure listener registered with MEMCG is notified of the event.



(http://jake.dothome.co.kr/wp-content/uploads/2019/09/vmpressure_event-1.png)

Calculating the VMpressure Level

vmpressure_calc_level()

mm/vmpressure.c

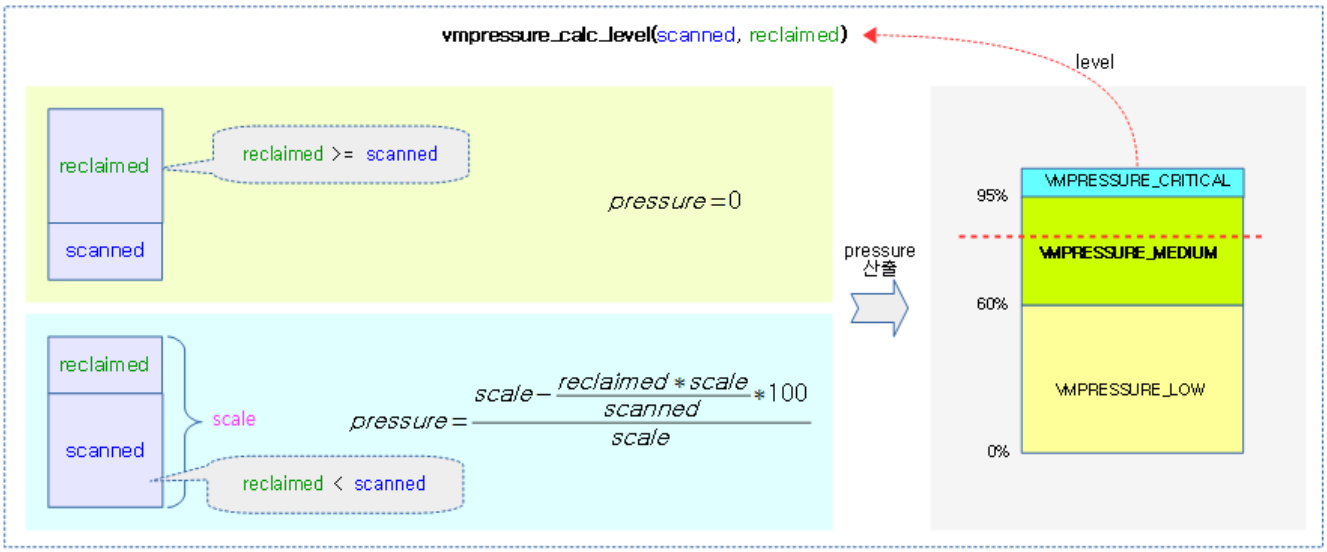
```
01 static enum vmpressure_levels vmpressure_calc_level(unsigned long scanne
   d,
02                                                     unsigned long reclai
   med)
03 {
04         unsigned long scale = scanned + reclaimed;
05         unsigned long pressure = 0;
06
07         /*
08          * reclaimed can be greater than scanned for things such as recl
   aimed
09          * slab pages. shrink_node() just adds reclaimed pages without a
10          * related increment to scanned pages.
11          */
12         if (reclaimed >= scanned)
13                 goto out;
14         /*
15          * We calculate the ratio (in percents) of how many pages were
16          * scanned vs. reclaimed in a given time frame (window). Note th
   at
17          * time is in VM reclaimer's "ticks", i.e. number of pages
18          * scanned. This makes it possible to set desired reaction time
19          * and serves as a ratelimit.
20          */
21         pressure = scale - (reclaimed * scale / scanned);
22         pressure = pressure * 100 / scale;
23
24 out:
25         pr_debug("%s: %3lu  (s: %lu  r: %lu)\n", __func__, pressure,
26                  scanned, reclaimed);
27
28         return vmpressure_level(pressure);
29 }
```

It calculates the pressure value according to the scanned and reclaimed ratio, and returns the level accordingly.

Let's check the pressure values and levels for the number of scanned and reclaimed pages, as shown in the following example.

- scanned=5, reclaimed=0
  - pressure=100%, level=critical
- scanned=5, reclaimed=1
  - pressure=66%, level=medium
- scanned=5, reclaimed=2
  - pressure=57%, level=low
- scanned=5, reclaimed=3
  - pressure=37%, level=low
- scanned=5, reclaimed=4
  - pressure=11%, level=low
- scanned=5, reclaimed=5
  - pressure=0%, level=low

The following figure shows the process of calculating the pressure value according to the scanned and reclaimed ratio and determining the level accordingly.



(http://jake.dothome.co.kr/wp-content/uploads/2016/07/vmpressure_calc_level-1.png)

vmpressure_level()

mm/vmpressure.c

```
1  static enum vmpressure_levels vmpressure_level(unsigned long pressure)
2  {
3          if (pressure >= vmpressure_level_critical)
4                  return VMPRESSURE_CRITICAL;
5          else if (pressure >= vmpressure_level_med)
6                  return VMPRESSURE_MEDIUM;
```

```
        return VMPRESSURE_LOW;
}
```

Returns the level according to the @pressure.

- critical
  - Default value of 95% or higher
- with
  - Default value of 60% or higher
- low
  - Others

Event Reception Program Demo

cgroup_event_listener

Running make in the tools/cgroup location will build the following source to create a cgroup_event_listener file:

tools/cgroup/cgroup_event_listener.c

```
#include <assert.h>
#include <err.h>
#include <errno.h>
#include <fcntl.h>
#include <libgen.h>
#include <limits.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include <sys/eventfd.h>

#define USAGE_STR "Usage: cgroup_event_listener <path-to-control-file> <args>"

int main(int argc, char **argv)
{
        int efd = -1;
        int cfd = -1;
        int event_control = -1;
        char event_control_path[PATH_MAX];
        char line[LINE_MAX];
        int ret;

        if (argc != 3)
                errx(1, "%s", USAGE_STR);

        cfd = open(argv[1], O_RDONLY);
        if (cfd == -1)
                err(1, "Cannot open %s", argv[1]);

        ret = snprintf(event_control_path, PATH_MAX, "%s/cgroup.event_control",
                       dirname(argv[1]));
        if (ret >= PATH_MAX)
                errx(1, "Path to cgroup.event_control is too long");

        event_control = open(event_control_path, O_WRONLY);
        if (event_control == -1)
```

```
38                  err(1, "Cannot open %s", event_control_path);
39
40          efd = eventfd(0, 0);
41          if (efd == -1)
42                  err(1, "eventfd() failed");
43
44          ret = snprintf(line, LINE_MAX, "%d %d %s", efd, cfd, argv[2]);
45          if (ret >= LINE_MAX)
46                  errx(1, "Arguments string is too long");
47
48          ret = write(event_control, line, strlen(line) + 1);
49          if (ret == -1)
50                  err(1, "Cannot write to cgroup.event_control");
51
52          while (1) {
53                  uint64_t result;
54
55                  ret = read(efd, &result, sizeof(result));
56                  if (ret == -1) {
57                          if (errno == EINTR)
58                                  continue;
59                          err(1, "Cannot read from eventfd");
60                  }
61                  assert(ret == sizeof(result));
62
63                  ret = access(event_control_path, W_OK);
64                  if ((ret == -1) && (errno == ENOENT)) {
65                          puts("The cgroup seems to have removed.");
66                          break;
67                  }
68
69                  if (ret == -1)
70                          err(1, "cgroup.event_control is not accessible a
   ny more");
71
72                  printf("%s %s: crossed\n", argv[1], argv[2]);
73          }
74
75          return 0;
76  }
```

How to use it

Allows you to receive events when the pressure level is medium, as follows:

- 참고: memcg: Add memory.pressure_level events (https://lwn.net/Articles/544652/) | LWN.net

```
# cd /sys/fs/cgroup/memory/
$ mkdir foo
$ cd foo
$ cgroup_event_listener memory.pressure_level medium &
$ echo 8000000 > memory.limit_in_bytes
$ echo 8000000 > memory.memsw.limit_in_bytes
$ echo $$ > tasks
$ dd if=/dev/zero | read x
```

PSI(Process Stall Information)

It was introduced in kernel v2018.4-rc20 in 1 to catch the level of pressure that user applications (such as lmkd on Android) are subjected to in memory reclamation operations.

- consultation
  - PSI – Pressure Stall Information (https://www.kernel.org/doc/Documentation/accounting/psi.txt) | kernel.org
  - psi: pressure stall information for CPU, memory, and IO (https://github.com/torvalds/linux/commit/eb414681d5a07d28d2ff90dc05f69ec6b232ebd2#diff-afd23506360c175bb0ef2e681113a7ba)
  - psi: cgroup support (https://github.com/torvalds/linux/commit/2ce7135adc9ad081aa3c49744144376ac74fea60#diff-afd23506360c175bb0ef2e681113a7ba)
  - psi: introduce psi monitor (https://github.com/torvalds/linux/commit/0e94682b73bfa6c44c98af7a26771c9c08c055d5#diff-afd23506360c175bb0ef2e681113a7ba)
  - Getting Started with PSI (https://facebookmicrosites.github.io/psi/docs/overview) | PSI

consultation

- Memory Resource Controller (https://lwn.net/Articles/432224/) (Documentation/cgroups/memory.txt) | LWN.net
- Low Memory Killer Daemon (https://source.android.com/devices/tech/perf/lmkd) | Android

---

LEAVE A COMMENT

Your email will not be published. Required fields are marked with *

Comments

name *

email *

Website

WRITE A COMMENT

< Rmap -1- (Reverse Mapping) (http://jake.dothome.co.kr/rmap-1/)

Swap -1- (Basic, initialization) > (http://jake.dothome.co.kr/swap-1/)

Munc Blog (2015 ~ 2024)