lowmemorykiller.c


```c
/* drivers/misc/lowmemorykiller.c
 *
 * The lowmemorykiller driver lets user-space specify a set of memory thresholds
 * where processes with a range of oom_score_adj values will get killed. Specify
 * the minimum oom_score_adj values in
 * /sys/module/lowmemorykiller/parameters/adj and the number of free pages in
 * /sys/module/lowmemorykiller/parameters/minfree. Both files take a comma
 * separated list of numbers in ascending order.
 *
 * For example, write "0,8" to /sys/module/lowmemorykiller/parameters/adj and
 * "1024,4096" to /sys/module/lowmemorykiller/parameters/minfree to kill
 * processes with a oom_score_adj value of 8 or higher when the free memory
 * drops below 4096 pages and kill processes with a oom_score_adj value of 0 or
 * higher when the free memory drops below 1024 pages.
 *
 * The driver considers memory used for caches to be free, but if a large
 * percentage of the cached memory is locked this can be very inaccurate
 * and processes may not get killed until the normal oom killer is triggered.
 *
 * Copyright (C) 2007-2008 Google, Inc.
 *
 * This software is licensed under the terms of the GNU General Public
 * License version 2, as published by the Free Software Foundation, and
 * may be copied, distributed, and modified under those terms.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 */

#define pr_fmt(fmt) KBUILD_MODNAME ": " fmt

#include <linux/init.h>
#include <linux/moduleparam.h>
#include <linux/kernel.h>
#include <linux/mm.h>
#include <linux/oom.h>
#include <linux/sched.h>
#include <linux/swap.h>
#include <linux/rcupdate.h>
#include <linux/profile.h>
#include <linux/notifier.h>
```

```
45
46  static u32 lowmem_debug_level = 1;
47  static short lowmem_adj[6] = {
48          0,
49          1,
50          6,
51          12,
52  };
53
54  static int lowmem_adj_size = 4;
55  static int lowmem_minfree[6] = {
56          3 * 512,        /* 6MB */
57          2 * 1024,       /* 8MB */
58          4 * 1024,       /* 16MB */
59          16 * 1024,      /* 64MB */
60  };
61
62  static int lowmem_minfree_size = 4;
63
64  static unsigned long lowmem_deathpending_timeout;
65
66  #define lowmem_print(level, x...)                       \
67          do {                                            \
68                  if (lowmem_debug_level >= (level))      \
69                          pr_info(x);                     \
70          } while (0)
71
72  static unsigned long lowmem_count(struct shrinker *s,
73                                    struct shrink_control *sc)
74  {
75          return global_page_state(NR_ACTIVE_ANON) +
76                  global_page_state(NR_ACTIVE_FILE) +
77                  global_page_state(NR_INACTIVE_ANON) +
78                  global_page_state(NR_INACTIVE_FILE);
79  }
80
81  static unsigned long lowmem_scan(struct shrinker *s, struct shrink_control *sc)
82  {
83          struct task_struct *tsk;
84          struct task_struct *selected = NULL;
85          unsigned long rem = 0;
86          int tasksize;
87          int i;
88          short min_score_adj = OOM_SCORE_ADJ_MAX + 1;
89          int minfree = 0;
90          int selected_tasksize = 0;
91          short selected_oom_score_adj;
92          int array_size = ARRAY_SIZE(lowmem_adj);
93          int other_free = global_page_state(NR_FREE_PAGES) - totalreserve_pages;
94          int other_file = global_page_state(NR_FILE_PAGES) -
95                                                  global_page_state(NR_SHMEM) -
96                                                  total_swapcache_pages();
97
```

```
        if (lowmem_adj_size < array_size)
                array_size = lowmem_adj_size;
        if (lowmem_minfree_size < array_size)
                array_size = lowmem_minfree_size;
        for (i = 0; i < array_size; i++) {
                minfree = lowmem_minfree[i];
                if (other_free < minfree && other_file < minfree) {
                        min_score_adj = lowmem_adj[i];
                        break;
                }
        }

        lowmem_print(3, "lowmem_scan %lu, %x, ofree %d %d, ma %hd\n",
                     sc->nr_to_scan, sc->gfp_mask, other_free,
                     other_file, min_score_adj);

        if (min_score_adj == OOM_SCORE_ADJ_MAX + 1) {
                lowmem_print(5, "lowmem_scan %lu, %x, return 0\n",
                             sc->nr_to_scan, sc->gfp_mask);
                return 0;
        }

        selected_oom_score_adj = min_score_adj;

        rcu_read_lock();
        for_each_process(tsk) {
                struct task_struct *p;
                short oom_score_adj;

                if (tsk->flags & PF_KTHREAD)
                        continue;

                p = find_lock_task_mm(tsk);
                if (!p)
                        continue;

                if (task_lmk_waiting(p) &&
                    time_before_eq(jiffies, lowmem_deathpending_timeout)) {
                        task_unlock(p);
                        rcu_read_unlock();
                        return 0;
                }
                oom_score_adj = p->signal->oom_score_adj;
                if (oom_score_adj < min_score_adj) {
                        task_unlock(p);
                        continue;
                }
                tasksize = get_mm_rss(p->mm);
                task_unlock(p);
                if (tasksize <= 0)
                        continue;
                if (selected) {
                        if (oom_score_adj < selected_oom_score_adj)
```

```
151                                 continue;
152                         if (oom_score_adj == selected_oom_score_adj &&
153                             tasksize <= selected_tasksize)
154                                 continue;
155                 }
156                 selected = p;
157                 selected_tasksize = tasksize;
158                 selected_oom_score_adj = oom_score_adj;
159                 lowmem_print(2, "select '%s' (%d), adj %hd, size %d, to kill\n",
160                              p->comm, p->pid, oom_score_adj, tasksize);
161         }
162         if (selected) {
163                 task_lock(selected);
164                 send_sig(SIGKILL, selected, 0);
165                 if (selected->mm)
166                         task_set_lmk_waiting(selected);
167                 task_unlock(selected);
168                 lowmem_print(1, "Killing '%s' (%d), adj %hd,\n"
169                                 "   to free %ldkB on behalf of '%s' (%d) because\n"
170                                 "   cache %ldkB is below limit %ldkB for oom_score_adj %hd\n"
171                                 "   Free memory is %ldkB above reserved\n",
172                              selected->comm, selected->pid,
173                              selected_oom_score_adj,
174                              selected_tasksize * (long)(PAGE_SIZE / 1024),
175                              current->comm, current->pid,
176                              other_file * (long)(PAGE_SIZE / 1024),
177                              minfree * (long)(PAGE_SIZE / 1024),
178                              min_score_adj,
179                              other_free * (long)(PAGE_SIZE / 1024));
180                 lowmem_deathpending_timeout = jiffies + HZ;
181                 rem += selected_tasksize;
182         }
183
184         lowmem_print(4, "lowmem_scan %lu, %x, return %lu\n",
185                      sc->nr_to_scan, sc->gfp_mask, rem);
186         rcu_read_unlock();
187         return rem;
188 }
189
190 static struct shrinker lowmem_shrinker = {
191         .scan_objects = lowmem_scan,
192         .count_objects = lowmem_count,
193         .seeks = DEFAULT_SEEKS * 16
194 };
195
196 static int __init lowmem_init(void)
197 {
198         register_shrinker(&lowmem_shrinker);
199         return 0;
200 }
201 device_initcall(lowmem_init);
202
203 /*
```

```
 * not really modular, but the easiest way to keep compat with existing
 * bootargs behaviour is to continue using module_param here.
 */
module_param_named(cost, lowmem_shrinker.seeks, int, S_IRUGO | S_IWUSR);
module_param_array_named(adj, lowmem_adj, short, &lowmem_adj_size,
                         S_IRUGO | S_IWUSR);
module_param_array_named(minfree, lowmem_minfree, uint, &lowmem_minfree_size,
                         S_IRUGO | S_IWUSR);
module_param_named(debug_level, lowmem_debug_level, uint, S_IRUGO | S_IWUSR);

```