Subject: [RFC][PATCH 5/7] Per-container OOM killer and page reclamation
Posted by xemul on Tue, 06 Mar 2007 15:01:05 GMT
View Forum Message <> Reply to Message

* container_try_to_free_pages() walks containers
  page list and tries to shrink pages. This is based
  on try_to_free_pages() and Co code.
  Called from core code when no resource left at the
  moment of page touching.

* container_out_of_memory() selects a process to be
  killed which mm_struct belongs to container in question.
  Called from core code when no resources left and no
  pages were reclaimed.

```
diff -upr linux-2.6.20.orig/mm/oom_kill.c linux-2.6.20-0/mm/oom_kill.c
--- linux-2.6.20.orig/mm/oom_kill.c 2007-03-06 13:33:28.000000000 +0300
+++ linux-2.6.20-0/mm/oom_kill.c 2007-03-06 13:33:28.000000000 +0300
@@ -24,6 +24,7 @@
 #include <linux/cpuset.h>
 #include <linux/module.h>
 #include <linux/notifier.h>
+#include <linux/rss_container.h>

 int sysctl_panic_on_oom;
 /* #define DEBUG */
@@ -47,7 +48,8 @@ int sysctl_panic_on_oom;
  *    of least surprise ... (be careful when you change it)
  */

-unsigned long badness(struct task_struct *p, unsigned long uptime)
+unsigned long badness(struct task_struct *p, unsigned long uptime,
+  struct rss_container *rss)
 {
  unsigned long points, cpu_time, run_time, s;
  struct mm_struct *mm;
@@ -60,6 +62,13 @@ unsigned long badness(struct task_struct
   return 0;
  }

+#ifdef CONFIG_RSS_CONTAINER
+ if (rss != NULL && mm->rss_container != rss) {
+  task_unlock(p);
+  return 0;
+ }
+#endif
+
  /*
```

```
   * The memory size of the process is the basis for the badness.
   */
@@ -200,7 +209,8 @@ static inline int constrained_alloc(stru
  *
  * (not docbooked, we don't want this one cluttering up the manual)
  */
-static struct task_struct *select_bad_process(unsigned long *ppoints)
+static struct task_struct *select_bad_process(unsigned long *ppoints,
+  struct rss_container *rss)
 {
  struct task_struct *g, *p;
  struct task_struct *chosen = NULL;
@@ -254,7 +264,7 @@ static struct task_struct *select_bad_pr
   if (p->oomkilladj == OOM_DISABLE)
    continue;

-  points = badness(p, uptime.tv_sec);
+  points = badness(p, uptime.tv_sec, rss);
   if (points > *ppoints || !chosen) {
    chosen = p;
    *ppoints = points;
@@ -435,7 +445,7 @@ retry:
    * Rambo mode: Shoot down a process and hope it solves whatever
    * issues we may have.
    */
-  p = select_bad_process(&points);
+  p = select_bad_process(&points, NULL);

   if (PTR_ERR(p) == -1UL)
    goto out;
@@ -464,3 +474,27 @@ out:
  if (!test_thread_flag(TIF_MEMDIE))
   schedule_timeout_uninterruptible(1);
 }
+
+#ifdef CONFIG_RSS_CONTAINER
+void container_out_of_memory(struct rss_container *rss)
+{
+ unsigned long points = 0;
+ struct task_struct *p;
+
+ container_lock();
+ read_lock(&tasklist_lock);
+retry:
+ p = select_bad_process(&points, rss);
+ if (PTR_ERR(p) == -1UL)
+  goto out;
+
```

```
+ if (!p)
+  p = current;
+
+ if (oom_kill_process(p, points, "Container out of memory"))
+  goto retry;
+out:
+ read_unlock(&tasklist_lock);
+ container_unlock();
+}
+#endif
diff -upr linux-2.6.20.orig/mm/vmscan.c linux-2.6.20-0/mm/vmscan.c
--- linux-2.6.20.orig/mm/vmscan.c 2007-02-04 21:44:54.000000000 +0300
+++ linux-2.6.20-0/mm/vmscan.c 2007-03-06 13:33:28.000000000 +0300
@@ -45,6 +45,8 @@

 #include "internal.h"

+#include <linux/rss_container.h>
+
 struct scan_control {
  /* Incremented by the number of inactive pages that were scanned */
  unsigned long nr_scanned;
@@ -1097,6 +1099,194 @@ out:
  return ret;
 }

+#ifdef CONFIG_RSS_CONTAINER
+/*
+ * These are containers' inactive and active pages shrinkers.
+ * Thes works like shrink_inactive_list() and shrink_active_list()
+ *
+ * Two main differences is that container_isolate_pages() is used to isolate
+ * pages, and that reclaim_mapped is considered to be 1 as hitting BC
+ * limit implies we have to shrink _mapped_ pages
+ */
+static unsigned long container_shrink_pages_inactive(unsigned long max_scan,
+  struct rss_container *rss, struct scan_control *sc)
+{
+ LIST_HEAD(page_list);
+ unsigned long nr_scanned = 0;
+ unsigned long nr_reclaimed = 0;
+
+ do {
+  struct page *page;
+  unsigned long nr_taken;
+  unsigned long nr_scan;
+  struct zone *z;
+
```

```
+  nr_taken = container_isolate_pages(sc->swap_cluster_max, rss,
+    &page_list, 0, &nr_scan);
+
+  nr_scanned += nr_scan;
+  nr_reclaimed += shrink_page_list(&page_list, sc);
+  if (nr_taken == 0)
+   goto done;
+
+  while (!list_empty(&page_list)) {
+   page = lru_to_page(&page_list);
+   z = page_zone(page);
+
+   spin_lock_irq(&z->lru_lock);
+   VM_BUG_ON(PageLRU(page));
+   SetPageLRU(page);
+   list_del(&page->lru);
+   if (PageActive(page))
+    add_page_to_active_list(z, page);
+   else
+    add_page_to_inactive_list(z, page);
+   spin_unlock_irq(&z->lru_lock);
+
+   put_page(page);
+  }
+   } while (nr_scanned < max_scan);
+done:
+ return nr_reclaimed;
+}
+
+static void container_shrink_pages_active(unsigned long nr_pages,
+  struct rss_container *rss, struct scan_control *sc)
+{
+ LIST_HEAD(l_hold);
+ LIST_HEAD(l_inactive);
+ LIST_HEAD(l_active);
+ struct page *page;
+ unsigned long nr_scanned;
+ unsigned long nr_deactivated = 0;
+ struct zone *z;
+
+ container_isolate_pages(nr_pages, rss, &l_hold, 1, &nr_scanned);
+
+ while (!list_empty(&l_hold)) {
+  cond_resched();
+  page = lru_to_page(&l_hold);
+  list_del(&page->lru);
+  if (page_mapped(page)) {
+   if ((total_swap_pages == 0 && PageAnon(page)) ||
```

```
+       page_referenced(page, 0)) {
+    list_add(&page->lru, &l_active);
+    continue;
+   }
+  }
+  nr_deactivated++;
+  list_add(&page->lru, &l_inactive);
+ }
+
+ while (!list_empty(&l_inactive)) {
+  page = lru_to_page(&l_inactive);
+  z = page_zone(page);
+
+  spin_lock_irq(&z->lru_lock);
+  VM_BUG_ON(PageLRU(page));
+  SetPageLRU(page);
+  VM_BUG_ON(!PageActive(page));
+  ClearPageActive(page);
+
+  list_move(&page->lru, &z->inactive_list);
+  z->nr_inactive++;
+  spin_unlock_irq(&z->lru_lock);
+
+  put_page(page);
+ }
+
+ while (!list_empty(&l_active)) {
+  page = lru_to_page(&l_active);
+  z = page_zone(page);
+
+  spin_lock_irq(&z->lru_lock);
+  VM_BUG_ON(PageLRU(page));
+  SetPageLRU(page);
+  VM_BUG_ON(!PageActive(page));
+  list_move(&page->lru, &z->active_list);
+  z->nr_active++;
+  spin_unlock_irq(&z->lru_lock);
+
+  put_page(page);
+ }
+}
+
+/*
+ * This is a reworked shrink_zone() routine - it scans active pages firts,
+ * then inactive and returns the number of pages reclaimed
+ */
+static unsigned long container_shrink_pages(int priority,
+  struct rss_container *rss, struct scan_control *sc)
```

```
+{
+ unsigned long nr_pages;
+ unsigned long nr_to_scan;
+ unsigned long nr_reclaimed = 0;
+
+ nr_pages = (container_nr_physpages(rss) >> priority) + 1;
+ if (nr_pages < sc->swap_cluster_max)
+  nr_pages = 0;
+
+ while (nr_pages) {
+  nr_to_scan = min(nr_pages, (unsigned long)sc->swap_cluster_max);
+  nr_pages -= nr_to_scan;
+  container_shrink_pages_active(nr_to_scan, rss, sc);
+ }
+
+ nr_pages = (container_nr_physpages(rss) >> priority) + 1;
+ if (nr_pages < sc->swap_cluster_max)
+  nr_pages = 0;
+
+ while (nr_pages) {
+  nr_to_scan = min(nr_pages, (unsigned long)sc->swap_cluster_max);
+  nr_pages -= nr_to_scan;
+  nr_reclaimed += container_shrink_pages_inactive(nr_to_scan, rss, sc);
+ }
+
+ throttle_vm_writeout();
+ return nr_reclaimed;
+}
+
+/*
+ * This functions works like try_to_free_pages() - it tries
+ * to shrink bc's pages with increasing priority
+ */
+unsigned long container_try_to_free_pages(struct rss_container *rss)
+{
+ int priority;
+ int ret = 0;
+ unsigned long total_scanned = 0;
+ unsigned long nr_reclaimed = 0;
+ struct scan_control sc = {
+  .gfp_mask = GFP_KERNEL,
+  .may_writepage = !laptop_mode,
+  .swap_cluster_max = SWAP_CLUSTER_MAX,
+  .may_swap = 1,
+  .swappiness = vm_swappiness,
+ };
+
+ for (priority = DEF_PRIORITY; priority >= 0; priority--) {
```

```
+  sc.nr_scanned = 0;
+  nr_reclaimed += container_shrink_pages(priority, rss, &sc);
+  total_scanned += sc.nr_scanned;
+  if (nr_reclaimed > 1) {
+   ret = 1;
+   goto out;
+  }
+
+  if (total_scanned > sc.swap_cluster_max +
+     sc.swap_cluster_max / 2) {
+   wakeup_pdflush(laptop_mode ? 0 : total_scanned);
+   sc.may_writepage = 1;
+  }
+
+  if (sc.nr_scanned && priority < DEF_PRIORITY - 2)
+   congestion_wait(WRITE, HZ/10);
+ }
+out:
+ return ret;
+}
+#endif
+
 /*
  * For kswapd, balance_pgdat() will work across all this node's zones until
  * they are all at pages_high.
```