Subject: [RFC][only for review] memory controller bacground reclaim [5/5]
Posted by KAMEZAWA Hiroyuki on Wed, 28 Nov 2007 08:57:13 GMT
View Forum Message <> Reply to Message

Create a daemon which does background page reclaim.

This daemon
  * starts when usage > high_watermark
  * stops when usage < low_watermark.

Because kthread_run() cannot be used when init_mem_cgroup is initialized(Sigh),
thread for init_mem_cgroup is invoked later by initcall.

Changes from YAMAMOTO-san's version
 * use kthread instead of workqueue.

Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
From: YAMAMOTO Takashi <yamamoto@valinux.co.jp>

 mm/memcontrol.c | 79
++
 1 file changed, 79 insertions(+)

Index: linux-2.6.24-rc3-mm1/mm/memcontrol.c
===
--- linux-2.6.24-rc3-mm1.orig/mm/memcontrol.c 2007-11-28 16:44:57.000000000 +0900
+++ linux-2.6.24-rc3-mm1/mm/memcontrol.c 2007-11-28 17:21:57.000000000 +0900
@@ -30,6 +30,8 @@
 #include <linux/spinlock.h>
 #include <linux/fs.h>
 #include <linux/seq_file.h>
+#include <linux/kthread.h>
+#include <linux/freezer.h>

 #include <asm/uaccess.h>

@@ -122,6 +124,13 @@
   */
  struct res_counter res;
  /*
+ * background reclaim
+ */
+ struct {
+ wait_queue_head_t waitq;
+ struct task_struct *thread;
+ } daemon;

```
+ /*
   * Per cgroup active and inactive list, similar to the
   * per zone LRU lists.
   */
@@ -401,6 +410,17 @@
  }
 }

+static void
+mem_cgroup_schedule_reclaim(struct mem_cgroup *mem)
+{
+ if (!unlikely(mem->daemon.thread))
+  return;
+ if (!waitqueue_active(&mem->daemon.waitq))
+  return;
+ wake_up_interruptible(&mem->daemon.waitq);
+}
+
+
 int task_in_mem_cgroup(struct task_struct *task, const struct mem_cgroup *mem)
 {
  int ret;
@@ -677,6 +697,8 @@
    mem_cgroup_out_of_memory(mem, GFP_KERNEL);
   goto free_pc;
  }
+ if (res_counter_above_high_watermark(&mem->res))
+  mem_cgroup_schedule_reclaim(mem);

  atomic_set(&pc->ref_cnt, 1);
  pc->mem_cgroup = mem;
@@ -832,6 +854,38 @@
 }

 /*
+ * Background page reclaim daeom for memory controller.
+ */
+
+static int mem_cgroup_reclaim_daemon(void *data)
+{
+ DEFINE_WAIT(wait);
+ struct mem_cgroup *mem = data;
+
+ css_get(&mem->css);
+ set_freezable();
+
+ while (!kthread_should_stop()) {
+  prepare_to_wait(&mem->daemon.waitq, &wait, TASK_INTERRUPTIBLE);
```

```
+
+  if (res_counter_below_low_watermark(&mem->res)) {
+   if (!kthread_should_stop()) {
+    schedule();
+    try_to_freeze();
+   }
+   finish_wait(&mem->daemon.waitq, &wait);
+   continue;
+  }
+  finish_wait(&mem->daemon.waitq, &wait);
+  try_to_free_mem_cgroup_pages(mem, GFP_HIGHUSER_MOVABLE);
+ }
+
+ css_put(&mem->css);
+
+ return 0;
+}
+
+/*
  * This routine traverse page_cgroup in given list and drop them all.
  * This routine ignores page_cgroup->ref_cnt.
  * *And* this routine doesn't reclaim page itself, just removes page_cgroup.
@@ -1045,6 +1100,7 @@
 {
  struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
  int ret;
+
  ret = mem_cgroup_force_empty(mem);
  if (!ret)
   ret = nbytes;
@@ -1188,6 +1244,16 @@

 static struct mem_cgroup init_mem_cgroup;

+static int __init mem_cgroup_reclaim_init(void)
+{
+ init_mem_cgroup.daemon.thread = kthread_run(mem_cgroup_reclaim_daemon,
+     &init_mem_cgroup, "memcontd");
+ if (IS_ERR(init_mem_cgroup.daemon.thread))
+  BUG();
+ return 0;
+}
+late_initcall(mem_cgroup_reclaim_init);
+
 static struct cgroup_subsys_state *
 mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
 {
@@ -1212,6 +1278,17 @@
```

```
   if (alloc_mem_cgroup_per_zone_info(mem, node))
    goto free_out;

+ /* Memory Reclaim Daemon per cgroup */
+ init_waitqueue_head(&mem->daemon.waitq);
+ if (mem != &init_mem_cgroup) {
+  /* Complicated...but we cannot call kthread create here..*/
+  /* init call will later assign kthread */
+  mem->daemon.thread = kthread_run(mem_cgroup_reclaim_daemon,
+     mem, "memcontd");
+  if (IS_ERR(mem->daemon.thread))
+   goto free_out;
+ }
+
  return &mem->css;
 free_out:
  for_each_node_state(node, N_POSSIBLE)
@@ -1226,6 +1303,7 @@
 {
  struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
  mem_cgroup_force_empty(mem);
+ kthread_stop(mem->daemon.thread);
 }

 static void mem_cgroup_destroy(struct cgroup_subsys *ss,
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers