
Subject: [PATCH v7 01/10] Basic kernel memory functionality for the Memory Controller

Posted by [Glauber Costa](#) on Tue, 29 Nov 2011 23:56:52 GMT

[View Forum Message](#) <> [Reply to Message](#)

This patch lays down the foundation for the kernel memory component of the Memory Controller.

As of today, I am only laying down the following files:

- * memory.independent_kmem_limit
- * memory.kmem.limit_in_bytes (currently ignored)
- * memory.kmem.usage_in_bytes (always zero)

Signed-off-by: Glauber Costa <glommer@parallels.com>

Reviewed-by: Kirill A. Shutemov <kirill@shutemov.name>

CC: Paul Menage <paul@paulmenage.org>

CC: Greg Thelen <gthelen@google.com>

```
Documentation/cgroups/memory.txt | 46 ++++++++
init/Kconfig                      | 14 +++++
mm/memcontrol.c                  | 103 ++++++++
3 files changed, 156 insertions(+), 7 deletions(-)
```

diff --git a/Documentation/cgroups/memory.txt b/Documentation/cgroups/memory.txt
index cc0ebc5..764781c 100644

--- a/Documentation/cgroups/memory.txt

+++ b/Documentation/cgroups/memory.txt

@@ -44,8 +44,9 @@ Features:

- oom-killer disable knob and oom-notifier
- Root cgroup has no limit controls.
- Kernel memory and Hugepages are not under control yet. We just manage
- pages on LRU. To add more controls, we have to take care of performance.
- + Hugepages is not under control yet. We just manage pages on LRU. To add more
- + controls, we have to take care of performance. Kernel memory support is work
- + in progress, and the current version provides basically functionality.

Brief summary of control files.

@@ -56,8 +57,11 @@ Brief summary of control files.

(See 5.5 for details)

memory.memsw.usage_in_bytes # show current res_counter usage for memory+Swap

(See 5.5 for details)

+ memory.kmem.usage_in_bytes # show current res_counter usage for kmem only.

+ (See 2.7 for details)

memory.limit_in_bytes # set/show limit of memory usage

memory.memsw.limit_in_bytes # set/show limit of memory+Swap usage

- + memory.kmem.limit_in_bytes # if allowed, set/show limit of kernel memory
- memory.failcnt # show the number of memory usage hits limits
- memory.memsw.failcnt # show the number of memory+Swap hits limits
- memory.max_usage_in_bytes # show max memory usage recorded
- @@ -72,6 +76,9 @@ Brief summary of control files.
- memory.oom_control # set/show oom controls.
- memory.numa_stat # show the number of memory usage per numa node
- + memory.independent_kmem_limit # select whether or not kernel memory limits are
- + independent of user limits
- +

1. History

The memory controller has a long history. A request for comments for the memory

@@ -255,6 +262,41 @@ When oom event notifier is registered, event will be delivered.

per-zone-per-cgroup LRU (cgroup's private LRU) is just guarded by zone->lru_lock, it has no lock of its own.

+2.7 Kernel Memory Extension (CONFIG_CGROUP_MEM_RES_CTLR_KMEM)

- +
- +With the Kernel memory extension, the Memory Controller is able to limit
- +the amount of kernel memory used by the system. Kernel memory is fundamentally
- +different than user memory, since it can't be swapped out, which makes it
- +possible to DoS the system by consuming too much of this precious resource.
- +
- +Some kernel memory resources may be accounted and limited separately from the
- +main "kmem" resource. For instance, a slab cache that is considered important
- +enough to be limited separately may have its own knobs.
- +
- +Kernel memory limits are not imposed for the root cgroup. Usage for the root
- +cgroup may or may not be accounted.
- +
- +Memory limits as specified by the standard Memory Controller may or may not
- +take kernel memory into consideration. This is achieved through the file
- +memory.independent_kmem_limit. A Value different than 0 will allow for kernel
- +memory to be controlled separately.
- +
- +When kernel memory limits are not independent, the limit values set in
- +memory.kmem files are ignored.
- +
- +Currently no soft limit is implemented for kernel memory. It is future work
- +to trigger slab reclaim when those limits are reached.
- +
- +CAUTION: As of this writing, the kmem extension may prevent tasks from moving
- +among cgroups. If a task has kmem accounting in a cgroup, the task cannot be
- +moved until the kmem resource is released. Also, until the resource is fully
- +released, the cgroup cannot be destroyed. So, please consider your use cases
- +and set kmem extension config option carefully.

+
+2.7.1 Current Kernel Memory resources accounted

+
+None

+
3. User Interface

0. Configuration

diff --git a/init/Kconfig b/init/Kconfig

index 43298f9..56558a2 100644

--- a/init/Kconfig

+++ b/init/Kconfig

@@ -689,6 +689,20 @@ config CGROUP_MEM_RES_CTLR_SWAP_ENABLED

For those who want to have the feature enabled by default should
select this option (if, for some reason, they need to disable it
then swapaccount=0 does the trick).

+config CGROUP_MEM_RES_CTLR_KMEM

+ bool "Memory Resource Controller Kernel Memory accounting (EXPERIMENTAL)"

+ depends on CGROUP_MEM_RES_CTLR && EXPERIMENTAL

+ default n

+ help

+ The Kernel Memory extension for Memory Resource Controller can limit
+ the amount of memory used by kernel objects in the system. Those are
+ fundamentally different from the entities handled by the standard
+ Memory Controller, which are page-based, and can be swapped. Users of
+ the kmem extension can use it to guarantee that no group of processes
+ will ever exhaust kernel resources alone.

+
+ WARNING: The current experimental implementation does not allow a
+ task to move among different cgroups with a kmem resource being held.

config CGROUP_PERF

bool "Enable perf_event per-cpu per-container group (cgroup) monitoring"

diff --git a/mm/memcontrol.c b/mm/memcontrol.c

index 6aff93c..3becb24 100644

--- a/mm/memcontrol.c

+++ b/mm/memcontrol.c

@@ -227,6 +227,10 @@ struct mem_cgroup {

*/

struct res_counter memsw;

/*

+ * the counter to account for kmem usage.

+ */

+ struct res_counter kmem;

+ /*

* Per cgroup active and inactive list, similar to the

* per zone LRU lists.

*/

```

@@ -277,6 +281,11 @@ struct mem_cgroup {
    */
    unsigned long move_charge_at_immigrate;
    /*
+ * Should kernel memory limits be stabilished independently
+ * from user memory ?
+ */
+ int kmem_independent_accounting;
+ /*
    * percpu counter.
    */
    struct mem_cgroup_stat_cpu *stat;
@@ -344,9 +353,14 @@ enum charge_type {
};

/* for encoding cft->private value on file */
#define _MEM (0)
#define _MEMSWAP (1)
#define _OOM_TYPE (2)
+
+enum mem_type {
+ _MEM = 0,
+ _MEMSWAP,
+ _OOM_TYPE,
+ _KMEM,
+};
+
#define MEMFILE_PRIVATE(x, val) (((x) << 16) | (val))
#define MEMFILE_TYPE(val) (((val) >> 16) & 0xffff)
#define MEMFILE_ATTR(val) ((val) & 0xffff)
@@ -3848,10 +3862,17 @@ static inline u64 mem_cgroup_usage(struct mem_cgroup *memcg,
bool swap)
    u64 val;

    if (!mem_cgroup_is_root(memcg)) {
+ val = 0;
+ #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (!memcg->kmem_independent_accounting)
+ val = res_counter_read_u64(&memcg->kmem, RES_USAGE);
+ #endif
        if (!swap)
- return res_counter_read_u64(&memcg->res, RES_USAGE);
+ val += res_counter_read_u64(&memcg->res, RES_USAGE);
        else
- return res_counter_read_u64(&memcg->memsw, RES_USAGE);
+ val += res_counter_read_u64(&memcg->memsw, RES_USAGE);
+
+ return val;

```

```

}

val = mem_cgroup_recursive_stat(memcg, MEM_CGROUP_STAT_CACHE);
@@ -3884,6 +3905,11 @@ static u64 mem_cgroup_read(struct cgroup *cont, struct cftype *cft)
    else
        val = res_counter_read_u64(&memcg->memsw, name);
    break;
#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ case _KMEM:
+ val = res_counter_read_u64(&memcg->kmem, name);
+ break;
#endif
    default:
        BUG();
        break;
@@ -4612,6 +4638,67 @@ static int mem_control_numa_stat_open(struct inode *unused, struct
file *file)
}
#endif /* CONFIG_NUMA */

#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+static u64 kmem_limit_independent_read(struct cgroup *cgroup, struct cftype *cft)
+{
+ return mem_cgroup_from_cont(cgroup)->kmem_independent_accounting;
+}
+
+static int kmem_limit_independent_write(struct cgroup *cgroup, struct cftype *cft,
+    u64 val)
+{
+ struct mem_cgroup *memcg = mem_cgroup_from_cont(cgroup);
+ struct mem_cgroup *parent = parent_mem_cgroup(memcg);
+
+ val = !!val;
+
+ if (parent && parent->use_hierarchy &&
+     (val != parent->kmem_independent_accounting))
+ return -EINVAL;
+ /*
+  * TODO: We need to handle the case in which we are doing
+  * independent kmem accounting as authorized by our parent,
+  * but then our parent changes its parameter.
+  */
+ cgroup_lock();
+ memcg->kmem_independent_accounting = val;
+ cgroup_unlock();
+ return 0;
+}
+static struct cftype kmem_cgroup_files[] = {

```



```

+ res_counter_init(&memcg->kmem, NULL);
}
memcg->last_scanned_child = 0;
memcg->last_scanned_node = MAX_NUMNODES;
@@ -4978,6 +5067,10 @@ static int mem_cgroup_populate(struct cgroup_subsys *ss,

    if (!ret)
        ret = register_memsw_files(cont, ss);
+
+ if (!ret)
+     ret = register_kmem_files(cont, ss);
+
    return ret;
}

```

--

1.7.6.4
