
Subject: [PATCH v5 2/8] socket: initial cgroup code.
Posted by [Glauber Costa](#) on Tue, 04 Oct 2011 12:17:54 GMT
[View Forum Message](#) <> [Reply to Message](#)

We aim to control the amount of kernel memory pinned at any time by tcp sockets. To lay the foundations for this work, this patch adds a pointer to the kmem_cgroup to the socket structure.

Signed-off-by: Glauber Costa <glommer@parallels.com>
Acked-by: Kirill A. Shutemov <kirill@shutemov.name>
Reviewed-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
CC: David S. Miller <davem@davemloft.net>
CC: Eric W. Biederman <ebiederm@xmission.com>

```
include/linux/memcontrol.h | 15 ++++++
include/net/sock.h         |  2 ++
mm/memcontrol.c            | 36 ++++++
net/core/sock.c           |  3 +++
4 files changed, 56 insertions(+), 0 deletions(-)
```

```
diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index 343bd76..88aea1b 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -376,5 +376,20 @@ mem_cgroup_print_bad_page(struct page *page)
 }
#endif
```

```
+#ifdef CONFIG_INET
+struct sock;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+void sock_update_memcg(struct sock *sk);
+void sock_release_memcg(struct sock *sk);
+
+#else
+static inline void sock_update_memcg(struct sock *sk)
+{
+}
+static inline void sock_release_memcg(struct sock *sk)
+{
+}
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
+#endif /* CONFIG_INET */
#endif /* _LINUX_MEMCONTROL_H */
```

```
diff --git a/include/net/sock.h b/include/net/sock.h
index 8e4062f..afe1467 100644
```

```

--- a/include/net/sock.h
+++ b/include/net/sock.h
@@ -228,6 +228,7 @@ struct sock_common {
    * @sk_security: used by security modules
    * @sk_mark: generic packet mark
    * @sk_classid: this socket's cgroup classid
+ * @sk_cgrp: this socket's kernel memory (kmem) cgroup
    * @sk_write_pending: a write to stream socket waits to start
    * @sk_state_change: callback to indicate change in the state of the sock
    * @sk_data_ready: callback to indicate there is data to be processed
@@ -339,6 +340,7 @@ struct sock {
#endif
    __u32 sk_mark;
    u32 sk_classid;
+ struct mem_cgroup *sk_cgrp;
    void (*sk_state_change)(struct sock *sk);
    void (*sk_data_ready)(struct sock *sk, int bytes);
    void (*sk_write_space)(struct sock *sk);
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 0871e3f..b1dcba9 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -296,6 +296,42 @@ struct mem_cgroup {
    spinlock_t pcp_counter_lock;
};

+/* Writing them here to avoid exposing memcg's inner layout */
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+#ifdef CONFIG_INET
+#include <net/sock.h>
+
+void sock_update_memcg(struct sock *sk)
+{
+ /* right now a socket spends its whole life in the same cgroup */
+ if (sk->sk_cgrp) {
+ WARN_ON(1);
+ return;
+ }
+
+ rcu_read_lock();
+ sk->sk_cgrp = mem_cgroup_from_task(current);
+
+ /*
+ * We don't need to protect against anything task-related, because
+ * we are basically stuck with the sock pointer that won't change,
+ * even if the task that originated the socket changes cgroups.
+ *
+ * What we do have to guarantee, is that the chain leading us to

```

