Subject: [PATCH v7 0/8] Request for inclusion: tcp memory buffers
Posted by Glauber Costa on Thu, 13 Oct 2011 13:09:34 GMT
View Forum Message <> Reply to Message

Hi Andrew,

This series was extensively reviewed over the past month, and after
all major comments were merged, I feel it is ready for inclusion when
the next merge window opens. Minor fixes will be provided if they
prove to be necessary.

You can see the most recent past discussions at:

https://lkml.org/lkml/2011/10/10/116
https://lkml.org/lkml/2011/10/4/116
https://lkml.org/lkml/2011/10/3/133

It basically lays the foundation for kernel memory limitation as a
general framework, and uses it to control the existent tcp memory pressure
thresholds in a per-cgroup way.

Follow up patches are expected for soft limits, which are not handled.

Please consider this for inclusion in your tree

Thanks,

Glauber Costa (8):
  Basic kernel memory functionality for the Memory Controller
  socket: initial cgroup code.
  foundations of per-cgroup memory pressure controlling.
  per-cgroup tcp buffers control
  per-netns ipv4 sysctl_tcp_mem
  tcp buffer limitation: per-cgroup limit
  Display current tcp memory allocation in kmem cgroup
  Disable task moving when using kernel memory accounting

 Documentation/cgroups/memory.txt | 38 ++++-
 crypto/af_alg.c | 8 +-
 include/linux/memcontrol.h | 49 +++++
 include/net/netns/ipv4.h | 1 +
 include/net/sock.h | 130 ++++++++++++-
 include/net/tcp.h | 30 +++-
 include/net/udp.h | 4 +-
 include/trace/events/sock.h | 10 +-
 init/Kconfig | 14 ++
 mm/memcontrol.c | 373 ++++++++++++++++++++++++++++++++++++++--
 net/core/sock.c | 104 ++++++++---

```
 net/decnet/af_decnet.c          |   22 ++-
 net/ipv4/proc.c                 |    7 +-
 net/ipv4/sysctl_net_ipv4.c      |   71 +++++++-
 net/ipv4/tcp.c                  |   60 ++++---
 net/ipv4/tcp_input.c            |   12 +-
 net/ipv4/tcp_ipv4.c             |   23 ++-
 net/ipv4/tcp_output.c           |    2 +-
 net/ipv4/tcp_timer.c            |    2 +-
 net/ipv4/udp.c                  |   21 ++-
 net/ipv6/tcp_ipv6.c             |   20 ++-
 net/ipv6/udp.c                  |    4 +-
 net/sctp/socket.c               |   37 +++-
 23 files changed, 910 insertions(+), 132 deletions(-)

--
1.7.6.4
```

Subject: [PATCH v7 1/8] Basic kernel memory functionality for the Memory Controller

Posted by Glauber Costa on Thu, 13 Oct 2011 13:09:35 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>
Reviewed-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
CC: Paul Menage <paul@paulmenage.org>
CC: Greg Thelen <gthelen@google.com>

```
 Documentation/cgroups/memory.txt |   36 +++++++++++++-
 init/Kconfig                     |   14 ++++++
 mm/memcontrol.c                  |   89 ++++++++++++++++++++++++++++++++++++--
 3 files changed, 132 insertions(+), 7 deletions(-)
```

diff --git a/Documentation/cgroups/memory.txt b/Documentation/cgroups/memory.txt
index 06eb6d9..0dafd70 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 +262,31 @@ 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.
+Kernel memory limits are not imposed for the root cgroup.
+
+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 extention 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 extention config option carefully.
+
 3. User Interface

 0. Configuration
diff --git a/init/Kconfig b/init/Kconfig
index d627783..b62b9e0 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 3508777..4f8a5bb 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -226,6 +226,10 @@ 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.
   */
@@ -276,6 +280,11 @@ 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;
@@ -343,9 +352,14 @@ 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)
@@ -3837,10 +3851,15 @@ static inline u64 mem_cgroup_usage(struct mem_cgroup *mem,
bool swap)
  u64 val;

  if (!mem_cgroup_is_root(mem)) {
+  val = 0;
+  if (!mem->kmem_independent_accounting)
+   val = res_counter_read_u64(&mem->kmem, RES_USAGE);
   if (!swap)
```

```
-   return res_counter_read_u64(&mem->res, RES_USAGE);
+   val += res_counter_read_u64(&mem->res, RES_USAGE);
    else
-   return res_counter_read_u64(&mem->memsw, RES_USAGE);
+   val += res_counter_read_u64(&mem->memsw, RES_USAGE);
+
+  return val;
   }

   val = mem_cgroup_recursive_stat(mem, MEM_CGROUP_STAT_CACHE);
@@ -3873,6 +3892,10 @@ static u64 mem_cgroup_read(struct cgroup *cont, struct cftype *cft)
    else
     val = res_counter_read_u64(&mem->memsw, name);
    break;
+ case _KMEM:
+  val = res_counter_read_u64(&mem->kmem, name);
+  break;
+
   default:
    BUG();
    break;
@@ -4603,6 +4626,20 @@ 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 *cont, struct cftype *cft)
+{
+ return mem_cgroup_from_cont(cont)->kmem_independent_accounting;
+}
+
+static int kmem_limit_independent_write(struct cgroup *cont, struct cftype *cft,
+     u64 val)
+{
+ mem_cgroup_from_cont(cont)->kmem_independent_accounting = !!val;
+ return 0;
+}
+#endif
+
 static struct cftype mem_cgroup_files[] = {
  {
   .name = "usage_in_bytes",
@@ -4718,6 +4755,42 @@ static int register_memsw_files(struct cgroup *cont, struct
cgroup_subsys *ss)
 }
 #endif
```

```
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+static struct cftype kmem_cgroup_files[] = {
+ {
+  .name = "independent_kmem_limit",
+  .read_u64 = kmem_limit_independent_read,
+  .write_u64 = kmem_limit_independent_write,
+ },
+ {
+  .name = "kmem.usage_in_bytes",
+  .private = MEMFILE_PRIVATE(_KMEM, RES_USAGE),
+  .read_u64 = mem_cgroup_read,
+ },
+ {
+  .name = "kmem.limit_in_bytes",
+  .private = MEMFILE_PRIVATE(_KMEM, RES_LIMIT),
+  .read_u64 = mem_cgroup_read,
+ },
+};
+
+static int register_kmem_files(struct cgroup *cont, struct cgroup_subsys *ss)
+{
+ int ret = 0;
+
+ ret = cgroup_add_files(cont, ss, kmem_cgroup_files,
+         ARRAY_SIZE(kmem_cgroup_files));
+ return ret;
+};
+
+#else
+static int register_kmem_files(struct cgroup *cont, struct cgroup_subsys *ss)
+{
+ return 0;
+}
+#endif
+
 static int alloc_mem_cgroup_per_zone_info(struct mem_cgroup *mem, int node)
 {
  struct mem_cgroup_per_node *pn;
@@ -4916,6 +4989,7 @@ mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
  if (parent && parent->use_hierarchy) {
   res_counter_init(&mem->res, &parent->res);
   res_counter_init(&mem->memsw, &parent->memsw);
+  res_counter_init(&mem->kmem, &parent->kmem);
   /*
    * We increment refcnt of the parent to ensure that we can
    * safely access it on res_counter_charge/uncharge.
@@ -4926,6 +5000,7 @@ mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
```

```
  } else {
   res_counter_init(&mem->res, NULL);
   res_counter_init(&mem->memsw, NULL);
+  res_counter_init(&mem->kmem, NULL);
  }
  mem->last_scanned_child = 0;
  mem->last_scanned_node = MAX_NUMNODES;
@@ -4969,6 +5044,10 @@ 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
```

Subject: [PATCH v7 2/8] socket: initial cgroup code.
Posted by Glauber Costa on Thu, 13 Oct 2011 13:09:36 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.fujtsu.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            |   37 ++++++++++++++++++++++++++++++++++++++++
 net/core/sock.c            |    3 +++
 4 files changed, 57 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 4f8a5bb..623841d 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -376,6 +376,43 @@ enum mem_type {
 #define MEM_CGROUP_RECLAIM_SOFT_BIT 0x2
 #define MEM_CGROUP_RECLAIM_SOFT  (1 << MEM_CGROUP_RECLAIM_SOFT_BIT)
```

```
+/* 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
+  * the top level won't change under our noses. Incrementing the
+  * reference count via cgroup_exclude_rmdir guarantees that.
+  */
+ cgroup_exclude_rmdir(mem_cgroup_css(sk->sk_cgrp));
+ rcu_read_unlock();
+}
+
+void sock_release_memcg(struct sock *sk)
+{
+ cgroup_release_and_wakeup_rmdir(mem_cgroup_css(sk->sk_cgrp));
+}
+#endif /* CONFIG_INET */
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
+
+
 static void mem_cgroup_get(struct mem_cgroup *mem);
 static void mem_cgroup_put(struct mem_cgroup *mem);
 static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *mem);
diff --git a/net/core/sock.c b/net/core/sock.c
index bc745d0..5426ba0 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -125,6 +125,7 @@
 #include <net/xfrm.h>
 #include <linux/ipsec.h>
 #include <net/cls_cgroup.h>
```

```
+#include <linux/memcontrol.h>

 #include <linux/filter.h>

@@ -1141,6 +1142,7 @@ struct sock *sk_alloc(struct net *net, int family, gfp_t priority,
   atomic_set(&sk->sk_wmem_alloc, 1);

   sock_update_classid(sk);
+  sock_update_memcg(sk);
  }

  return sk;
@@ -1172,6 +1174,7 @@ static void __sk_free(struct sock *sk)
   put_cred(sk->sk_peer_cred);
  put_pid(sk->sk_peer_pid);
  put_net(sock_net(sk));
+ sock_release_memcg(sk);
  sk_prot_free(sk->sk_prot_creator, sk);
 }

--
1.7.6.4
```

Subject: [PATCH v7 3/8] foundations of per-cgroup memory pressure controlling.
Posted by Glauber Costa on Thu, 13 Oct 2011 13:09:37 GMT
View Forum Message <> Reply to Message

This patch converts struct sock fields memory_pressure,
memory_allocated, sockets_allocated, and sysctl_mem (now prot_mem)
to function pointers, receiving a struct mem_cgroup parameter.

enter_memory_pressure is kept the same, since all its callers
have socket a context, and the kmem_cgroup can be derived from
the socket itself.

To keep things working, the patch convert all users of those fields
to use acessor functions.

In my benchmarks I didn't see a significant performance difference
with this patch applied compared to a baseline (around 1 % diff, thus
inside error margin).

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

```
 crypto/af_alg.c            |    8 +++-
 include/linux/memcontrol.h  |   22 ++++++++-
 include/net/sock.h          |  114 +++++++++++++++++++++++++++++++++++++++++++--
 include/net/tcp.h           |   12 +++--
 include/net/udp.h           |    4 +-
 include/trace/events/sock.h |   10 ++--
 mm/memcontrol.c             |   19 ++++++-
 net/core/sock.c             |   62 ++++++++++++++---------
 net/decnet/af_decnet.c      |   22 +++++++-
 net/ipv4/proc.c             |    7 ++-
 net/ipv4/tcp.c              |   28 +++++++++-
 net/ipv4/tcp_input.c        |   12 ++--
 net/ipv4/tcp_ipv4.c         |   12 ++--
 net/ipv4/tcp_output.c       |    2 +-
 net/ipv4/tcp_timer.c        |    2 +-
 net/ipv4/udp.c              |   21 ++++++--
 net/ipv6/tcp_ipv6.c         |   10 ++--
 net/ipv6/udp.c              |    4 +-
 net/sctp/socket.c           |   37 +++++++++++---
 19 files changed, 320 insertions(+), 88 deletions(-)

diff --git a/crypto/af_alg.c b/crypto/af_alg.c
index ac33d5f..09cdf11 100644
--- a/crypto/af_alg.c
+++ b/crypto/af_alg.c
@@ -29,10 +29,16 @@ struct alg_type_list {

 static atomic_long_t alg_memory_allocated;

+static long memory_allocated_alg(struct mem_cgroup *memcg, long val,
+     int *parent_status)
+{
+ return atomic_long_add_return(val, &alg_memory_allocated);
+}
+
 static struct proto alg_proto = {
  .name   = "ALG",
  .owner   = THIS_MODULE,
- .memory_allocated = &alg_memory_allocated,
+ .mem_allocated_add = memory_allocated_alg,
  .obj_size  = sizeof(struct alg_sock),
 };

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index 88aea1b..99a8ba2 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -361,6 +361,10 @@ static inline
```

```
 void mem_cgroup_count_vm_event(struct mm_struct *mm, enum vm_event_item idx)
 {
 }
+static inline struct mem_cgroup *mem_cgroup_from_task(struct task_struct *p)
+{
+ return NULL;
+}
 #endif /* CONFIG_CGROUP_MEM_CONT */

 #if !defined(CONFIG_CGROUP_MEM_RES_CTLR) || !defined(CONFIG_DEBUG_VM)
@@ -377,12 +381,28 @@ mem_cgroup_print_bad_page(struct page *page)
 #endif

 #ifdef CONFIG_INET
+enum {
+ UNDER_LIMIT,
+ OVER_LIMIT,
+};
+
 struct sock;
+struct proto;
 #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
 void sock_update_memcg(struct sock *sk);
 void sock_release_memcg(struct sock *sk);
-
+void memcg_sockets_allocated_dec(struct mem_cgroup *memcg, struct proto *prot);
+void memcg_sockets_allocated_inc(struct mem_cgroup *memcg, struct proto *prot);
 #else
+/* memcontrol includes sockets.h, that includes memcontrol.h ... */
+static inline void memcg_sockets_allocated_dec(struct mem_cgroup *memcg,
+           struct proto *prot)
+{
+}
+static inline void memcg_sockets_allocated_inc(struct mem_cgroup *memcg,
+           struct proto *prot)
+{
+}
 static inline void sock_update_memcg(struct sock *sk)
 {
 }
diff --git a/include/net/sock.h b/include/net/sock.h
index afe1467..163f87b 100644
--- a/include/net/sock.h
+++ b/include/net/sock.h
@@ -54,6 +54,7 @@
 #include <linux/security.h>
 #include <linux/slab.h>
 #include <linux/uaccess.h>
```

```
+#include <linux/cgroup.h>

 #include <linux/filter.h>
 #include <linux/rculist_nulls.h>
@@ -168,6 +169,8 @@ struct sock_common {
  /* public: */
 };

+struct mem_cgroup;
+
 /**
   * struct sock - network layer representation of sockets
   * @__sk_common: shared layout with inet_timewait_sock
@@ -786,18 +789,36 @@ struct proto {
  unsigned int  inuse_idx;
 #endif

+ /*
+  * per-cgroup memory tracking:
+  *
+  * The following functions track memory consumption of network buffers
+  * by cgroup (kmem_cgroup) for the current protocol. As of the rest
+  * of the fields in this structure, not all protocols are required
+  * to implement them. Protocols that don't want to do per-cgroup
+  * memory pressure management, can just assume the root cgroup is used.
+  *
+  */
  /* Memory pressure */
  void   (*enter_memory_pressure)(struct sock *sk);
- atomic_long_t  *memory_allocated; /* Current allocated memory. */
- struct percpu_counter *sockets_allocated; /* Current number of sockets. */
  /*
-  * Pressure flag: try to collapse.
+  * Add a value in pages to the current memory allocation,
+  * and return the current value.
+  */
+ long   (*mem_allocated_add)(struct mem_cgroup *memcg,
+         long val, int *parent_status);
+ /* Pointer to the current number of sockets in this cgroup. */
+ struct percpu_counter *(*sockets_allocated)(const struct mem_cgroup *memcg);
+ /*
+  * Per cgroup pointer to the pressure flag: try to collapse.
   * Technical note: it is used by multiple contexts non atomically.
   * All the __sk_mem_schedule() is of this nature: accounting
   * is strict, actions are advisory and have some latency.
   */
- int   *memory_pressure;
- long   *sysctl_mem;
```

```
+ int   *(*memory_pressure)(const struct mem_cgroup *memcg);
+ /* Pointer to the per-cgroup version of the the sysctl_mem field */
+ long   *(*prot_mem)(const struct mem_cgroup *memcg);
+
  int   *sysctl_wmem;
  int   *sysctl_rmem;
  int   max_header;
@@ -856,6 +877,87 @@ static inline void sk_refcnt_debug_release(const struct sock *sk)
 #define sk_refcnt_debug_release(sk) do { } while (0)
 #endif /* SOCK_REFCNT_DEBUG */

+#include <linux/memcontrol.h>
+static inline int *sk_memory_pressure(struct sock *sk)
+{
+ int *ret = NULL;
+ if (sk->sk_prot->memory_pressure)
+  ret = sk->sk_prot->memory_pressure(sk->sk_cgrp);
+ return ret;
+}
+
+static inline long sk_prot_mem(struct sock *sk, int index)
+{
+ long *prot = sk->sk_prot->prot_mem(sk->sk_cgrp);
+ return prot[index];
+}
+
+static inline long
+sk_memory_allocated(struct sock *sk)
+{
+ struct proto *prot = sk->sk_prot;
+ struct mem_cgroup *cg = sk->sk_cgrp;
+
+ return prot->mem_allocated_add(cg, 0, NULL);
+}
+
+static inline long
+sk_memory_allocated_add(struct sock *sk, int amt, int *parent_status)
+{
+ struct proto *prot = sk->sk_prot;
+ struct mem_cgroup *cg = sk->sk_cgrp;
+
+ return prot->mem_allocated_add(cg, amt, parent_status);
+}
+
+static inline void
+sk_memory_allocated_sub(struct sock *sk, int amt, int parent_status)
+{
+ struct proto *prot = sk->sk_prot;
```

```
+ struct mem_cgroup *cg = sk->sk_cgrp;
+
+ prot->mem_allocated_add(cg, -amt, &parent_status);
+}
+
+static inline void sk_sockets_allocated_dec(struct sock *sk)
+{
+ struct proto *prot = sk->sk_prot;
+ struct mem_cgroup *cg = sk->sk_cgrp;
+
+ percpu_counter_dec(prot->sockets_allocated(cg));
+ memcg_sockets_allocated_dec(cg, prot);
+}
+
+static inline void sk_sockets_allocated_inc(struct sock *sk)
+{
+ struct proto *prot = sk->sk_prot;
+ struct mem_cgroup *cg = sk->sk_cgrp;
+
+ percpu_counter_inc(prot->sockets_allocated(cg));
+ memcg_sockets_allocated_inc(cg, prot);
+}
+
+static inline int
+sk_sockets_allocated_read_positive(struct sock *sk)
+{
+ struct proto *prot = sk->sk_prot;
+ struct mem_cgroup *cg = sk->sk_cgrp;
+
+ return percpu_counter_sum_positive(prot->sockets_allocated(cg));
+}
+
+static inline int
+kcg_sockets_allocated_sum_positive(struct proto *prot, struct mem_cgroup *cg)
+{
+ return percpu_counter_sum_positive(prot->sockets_allocated(cg));
+}
+
+static inline long
+kcg_memory_allocated(struct proto *prot, struct mem_cgroup *cg)
+{
+ return prot->mem_allocated_add(cg, 0, NULL);
+}
+

 #ifdef CONFIG_PROC_FS
 /* Called with local bh disabled */
@@ -952,7 +1054,7 @@ static inline int sk_mem_pages(int amt)
```

```
 static inline int sk_has_account(struct sock *sk)
 {
  /* return true if protocol supports memory accounting */
- return !!sk->sk_prot->memory_allocated;
+ return !!sk->sk_prot->mem_allocated_add;
 }

 static inline int sk_wmem_schedule(struct sock *sk, int size)
diff --git a/include/net/tcp.h b/include/net/tcp.h
index acc620a..eac7bf6 100644
--- a/include/net/tcp.h
+++ b/include/net/tcp.h
@@ -45,6 +45,7 @@
 #include <net/dst.h>

 #include <linux/seq_file.h>
+#include <linux/memcontrol.h>

 extern struct inet_hashinfo tcp_hashinfo;

@@ -253,9 +254,12 @@ extern int sysctl_tcp_cookie_size;
 extern int sysctl_tcp_thin_linear_timeouts;
 extern int sysctl_tcp_thin_dupack;

-extern atomic_long_t tcp_memory_allocated;
-extern struct percpu_counter tcp_sockets_allocated;
-extern int tcp_memory_pressure;
+struct mem_cgroup;
+extern long *tcp_sysctl_mem(const struct mem_cgroup *memcg);
+struct percpu_counter *sockets_allocated_tcp(const struct mem_cgroup *memcg);
+int *memory_pressure_tcp(const struct mem_cgroup *memcg);
+long memory_allocated_tcp_add(struct mem_cgroup *memcg, long val,
+         int *parent_status);

 /*
  * The next routines deal with comparing 32 bit unsigned ints
@@ -286,7 +290,7 @@ static inline bool tcp_too_many_orphans(struct sock *sk, int shift)
  }

  if (sk->sk_wmem_queued > SOCK_MIN_SNDBUF &&
-     atomic_long_read(&tcp_memory_allocated) > sysctl_tcp_mem[2])
+     sk_memory_allocated(sk) > sk_prot_mem(sk, 2))
   return true;
  return false;
 }
diff --git a/include/net/udp.h b/include/net/udp.h
index 67ea6fc..eecd727 100644
--- a/include/net/udp.h
```

```
+++ b/include/net/udp.h
@@ -105,7 +105,9 @@ static inline struct udp_hslot *udp_hashslot2(struct udp_table *table,

 extern struct proto udp_prot;

-extern atomic_long_t udp_memory_allocated;
+long memory_allocated_udp_add(struct mem_cgroup *memcg, long val,
+         int *parent_status);
+long *udp_sysctl_mem(const struct mem_cgroup *memcg);

 /* sysctl variables for udp */
 extern long sysctl_udp_mem[3];
diff --git a/include/trace/events/sock.h b/include/trace/events/sock.h
index 779abb9..12a6083 100644
--- a/include/trace/events/sock.h
+++ b/include/trace/events/sock.h
@@ -37,7 +37,7 @@ TRACE_EVENT(sock_exceed_buf_limit,

  TP_STRUCT__entry(
   __array(char, name, 32)
-  __field(long *, sysctl_mem)
+  __field(long *, prot_mem)
   __field(long, allocated)
   __field(int, sysctl_rmem)
   __field(int, rmem_alloc)
@@ -45,7 +45,7 @@ TRACE_EVENT(sock_exceed_buf_limit,

  TP_fast_assign(
   strncpy(__entry->name, prot->name, 32);
-  __entry->sysctl_mem = prot->sysctl_mem;
+  __entry->prot_mem = sk->sk_prot->prot_mem(sk->sk_cgrp);
   __entry->allocated = allocated;
   __entry->sysctl_rmem = prot->sysctl_rmem[0];
   __entry->rmem_alloc = atomic_read(&sk->sk_rmem_alloc);
@@ -54,9 +54,9 @@ TRACE_EVENT(sock_exceed_buf_limit,
  TP_printk("proto:%s sysctl_mem=%ld,%ld,%ld allocated=%ld "
   "sysctl_rmem=%d rmem_alloc=%d",
   __entry->name,
-  __entry->sysctl_mem[0],
-  __entry->sysctl_mem[1],
-  __entry->sysctl_mem[2],
+  __entry->prot_mem[0],
+  __entry->prot_mem[1],
+  __entry->prot_mem[2],
   __entry->allocated,
   __entry->sysctl_rmem,
   __entry->rmem_alloc)
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
```

```
index 623841d..4e71fd8 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -376,6 +376,7 @@ enum mem_type {
 #define MEM_CGROUP_RECLAIM_SOFT_BIT 0x2
 #define MEM_CGROUP_RECLAIM_SOFT  (1 << MEM_CGROUP_RECLAIM_SOFT_BIT)

+static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *memcg);
 /* Writing them here to avoid exposing memcg's inner layout */
 #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
 #ifdef CONFIG_INET
@@ -409,13 +410,27 @@ void sock_release_memcg(struct sock *sk)
 {
  cgroup_release_and_wakeup_rmdir(mem_cgroup_css(sk->sk_cgrp));
 }
+
+void memcg_sockets_allocated_dec(struct mem_cgroup *memcg, struct proto *prot)
+{
+ memcg = parent_mem_cgroup(memcg);
+ for (; memcg; memcg = parent_mem_cgroup(memcg))
+  percpu_counter_dec(prot->sockets_allocated(memcg));
+}
+EXPORT_SYMBOL(memcg_sockets_allocated_dec);
+
+void memcg_sockets_allocated_inc(struct mem_cgroup *memcg, struct proto *prot)
+{
+ memcg = parent_mem_cgroup(memcg);
+ for (; memcg; memcg = parent_mem_cgroup(memcg))
+  percpu_counter_inc(prot->sockets_allocated(memcg));
+}
+EXPORT_SYMBOL(memcg_sockets_allocated_inc);
 #endif /* CONFIG_INET */
 #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */

-
 static void mem_cgroup_get(struct mem_cgroup *mem);
 static void mem_cgroup_put(struct mem_cgroup *mem);
-static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *mem);
 static void drain_all_stock_async(struct mem_cgroup *mem);

 static struct mem_cgroup_per_zone *
diff --git a/net/core/sock.c b/net/core/sock.c
index 5426ba0..22ef143 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -1293,7 +1293,7 @@ struct sock *sk_clone(const struct sock *sk, const gfp_t priority)
   newsk->sk_wq = NULL;
```

```
   if (newsk->sk_prot->sockets_allocated)
-   percpu_counter_inc(newsk->sk_prot->sockets_allocated);
+   sk_sockets_allocated_inc(newsk);

   if (sock_flag(newsk, SOCK_TIMESTAMP) ||
       sock_flag(newsk, SOCK_TIMESTAMPING_RX_SOFTWARE))
@@ -1684,30 +1684,33 @@ int __sk_mem_schedule(struct sock *sk, int size, int kind)
  struct proto *prot = sk->sk_prot;
  int amt = sk_mem_pages(size);
  long allocated;
+ int *memory_pressure;
+ int parent_status = UNDER_LIMIT;

  sk->sk_forward_alloc += amt * SK_MEM_QUANTUM;
- allocated = atomic_long_add_return(amt, prot->memory_allocated);
+
+ memory_pressure = sk_memory_pressure(sk);
+ allocated = sk_memory_allocated_add(sk, amt, &parent_status);
+
+ /* Over hard limit (we, or our parents) */
+ if ((parent_status == OVER_LIMIT) || (allocated > sk_prot_mem(sk, 2)))
+  goto suppress_allocation;

  /* Under limit. */
- if (allocated <= prot->sysctl_mem[0]) {
-  if (prot->memory_pressure && *prot->memory_pressure)
-   *prot->memory_pressure = 0;
-  return 1;
- }
+ if (allocated <= sk_prot_mem(sk, 0))
+  if (memory_pressure && *memory_pressure)
+   *memory_pressure = 0;

  /* Under pressure. */
- if (allocated > prot->sysctl_mem[1])
+ if (allocated > sk_prot_mem(sk, 1))
   if (prot->enter_memory_pressure)
    prot->enter_memory_pressure(sk);

- /* Over hard limit. */
- if (allocated > prot->sysctl_mem[2])
-  goto suppress_allocation;
-
  /* guarantee minimum buffer size under pressure */
  if (kind == SK_MEM_RECV) {
   if (atomic_read(&sk->sk_rmem_alloc) < prot->sysctl_rmem[0])
    return 1;
+
```

```
  } else { /* SK_MEM_SEND */
   if (sk->sk_type == SOCK_STREAM) {
    if (sk->sk_wmem_queued < prot->sysctl_wmem[0])
@@ -1717,13 +1720,13 @@ int __sk_mem_schedule(struct sock *sk, int size, int kind)
     return 1;
  }

- if (prot->memory_pressure) {
+ if (memory_pressure) {
   int alloc;

-  if (!*prot->memory_pressure)
+  if (!*memory_pressure)
    return 1;
-  alloc = percpu_counter_read_positive(prot->sockets_allocated);
-  if (prot->sysctl_mem[2] > alloc *
+  alloc = sk_sockets_allocated_read_positive(sk);
+  if (sk_prot_mem(sk, 2) > alloc *
       sk_mem_pages(sk->sk_wmem_queued +
      atomic_read(&sk->sk_rmem_alloc) +
      sk->sk_forward_alloc))
@@ -1746,7 +1749,9 @@ suppress_allocation:

  /* Alas. Undo changes. */
  sk->sk_forward_alloc -= amt * SK_MEM_QUANTUM;
- atomic_long_sub(amt, prot->memory_allocated);
+
+ sk_memory_allocated_sub(sk, amt, parent_status);
+
  return 0;
 }
 EXPORT_SYMBOL(__sk_mem_schedule);
@@ -1757,15 +1762,15 @@ EXPORT_SYMBOL(__sk_mem_schedule);
  */
 void __sk_mem_reclaim(struct sock *sk)
 {
- struct proto *prot = sk->sk_prot;
+ int *memory_pressure = sk_memory_pressure(sk);

- atomic_long_sub(sk->sk_forward_alloc >> SK_MEM_QUANTUM_SHIFT,
-     prot->memory_allocated);
+ sk_memory_allocated_sub(sk,
+     sk->sk_forward_alloc >> SK_MEM_QUANTUM_SHIFT, 0);
  sk->sk_forward_alloc &= SK_MEM_QUANTUM - 1;

- if (prot->memory_pressure && *prot->memory_pressure &&
-     (atomic_long_read(prot->memory_allocated) < prot->sysctl_mem[0]))
-  *prot->memory_pressure = 0;
```

```
+ if (memory_pressure && *memory_pressure &&
+     (sk_memory_allocated(sk) < sk_prot_mem(sk, 0)))
+  *memory_pressure = 0;
 }
 EXPORT_SYMBOL(__sk_mem_reclaim);

@@ -2484,13 +2489,20 @@ static char proto_method_implemented(const void *method)

 static void proto_seq_printf(struct seq_file *seq, struct proto *proto)
 {
+ struct mem_cgroup *cg = mem_cgroup_from_task(current);
+ int *memory_pressure = NULL;
+
+ if (proto->memory_pressure)
+  memory_pressure = proto->memory_pressure(cg);
+
  seq_printf(seq, "%-9s %4u %6d  %6ld   %-3s %6u   %-3s  %-10s "
    "%2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c
%2c\n",
      proto->name,
      proto->obj_size,
      sock_prot_inuse_get(seq_file_net(seq), proto),
-     proto->memory_allocated != NULL ? atomic_long_read(proto->memory_allocated) : -1L,
-     proto->memory_pressure != NULL ? *proto->memory_pressure ? "yes" : "no" : "NI",
+     proto->mem_allocated_add != NULL ?
+   kcg_memory_allocated(proto, cg) : -1L,
+     memory_pressure != NULL ? *memory_pressure ? "yes" : "no" : "NI",
      proto->max_header,
      proto->slab == NULL ? "no" : "yes",
      module_name(proto->owner),
diff --git a/net/decnet/af_decnet.c b/net/decnet/af_decnet.c
index 19acd00..724ac73 100644
--- a/net/decnet/af_decnet.c
+++ b/net/decnet/af_decnet.c
@@ -458,13 +458,29 @@ static void dn_enter_memory_pressure(struct sock *sk)
  }
 }

+static long memory_allocated_dn_add(struct mem_cgroup *memcg,
+           long val, int *parent_status)
+{
+ return atomic_long_add_return(val, &decnet_memory_allocated);
+}
+
+static int *memory_pressure_dn(const struct mem_cgroup *memcg)
+{
+ return &dn_memory_pressure;
+}
```

```
+
+static long *dn_sysctl_mem(const struct mem_cgroup *memcg)
+{
+ return sysctl_decnet_mem;
+}
+
 static struct proto dn_proto = {
  .name   = "NSP",
  .owner   = THIS_MODULE,
  .enter_memory_pressure = dn_enter_memory_pressure,
- .memory_pressure = &dn_memory_pressure,
- .memory_allocated = &decnet_memory_allocated,
- .sysctl_mem  = sysctl_decnet_mem,
+ .memory_pressure = memory_pressure_dn,
+ .mem_allocated_add = memory_allocated_dn_add,
+ .prot_mem  = dn_sysctl_mem,
  .sysctl_wmem  = sysctl_decnet_wmem,
  .sysctl_rmem  = sysctl_decnet_rmem,
  .max_header  = DN_MAX_NSP_DATA_HEADER + 64,
diff --git a/net/ipv4/proc.c b/net/ipv4/proc.c
index 4bfad5d..535456d 100644
--- a/net/ipv4/proc.c
+++ b/net/ipv4/proc.c
@@ -52,20 +52,21 @@ static int sockstat_seq_show(struct seq_file *seq, void *v)
 {
  struct net *net = seq->private;
  int orphans, sockets;
+ struct mem_cgroup *cg = mem_cgroup_from_task(current);

  local_bh_disable();
  orphans = percpu_counter_sum_positive(&tcp_orphan_count);
- sockets = percpu_counter_sum_positive(&tcp_sockets_allocated);
+ sockets = kcg_sockets_allocated_sum_positive(&tcp_prot, cg);
  local_bh_enable();

  socket_seq_show(seq);
  seq_printf(seq, "TCP: inuse %d orphan %d tw %d alloc %d mem %ld\n",
      sock_prot_inuse_get(net, &tcp_prot), orphans,
      tcp_death_row.tw_count, sockets,
-     atomic_long_read(&tcp_memory_allocated));
+     kcg_memory_allocated(&tcp_prot, cg));
  seq_printf(seq, "UDP: inuse %d mem %ld\n",
      sock_prot_inuse_get(net, &udp_prot),
-     atomic_long_read(&udp_memory_allocated));
+     kcg_memory_allocated(&udp_prot, cg));
  seq_printf(seq, "UDPLITE: inuse %d\n",
      sock_prot_inuse_get(net, &udplite_prot));
  seq_printf(seq, "RAW: inuse %d\n",
```

```
diff --git a/net/ipv4/tcp.c b/net/ipv4/tcp.c
index 46febca..dc8f01e 100644
--- a/net/ipv4/tcp.c
+++ b/net/ipv4/tcp.c
@@ -291,13 +291,11 @@ EXPORT_SYMBOL(sysctl_tcp_rmem);
 EXPORT_SYMBOL(sysctl_tcp_wmem);

 atomic_long_t tcp_memory_allocated; /* Current allocated memory. */
-EXPORT_SYMBOL(tcp_memory_allocated);

 /*
  * Current number of TCP sockets.
  */
 struct percpu_counter tcp_sockets_allocated;
-EXPORT_SYMBOL(tcp_sockets_allocated);

 /*
  * TCP splice context
@@ -315,7 +313,18 @@ struct tcp_splice_state {
  * is strict, actions are advisory and have some latency.
  */
 int tcp_memory_pressure __read_mostly;
-EXPORT_SYMBOL(tcp_memory_pressure);
+
+int *memory_pressure_tcp(const struct mem_cgroup *memcg)
+{
+ return &tcp_memory_pressure;
+}
+EXPORT_SYMBOL(memory_pressure_tcp);
+
+struct percpu_counter *sockets_allocated_tcp(const struct mem_cgroup *memcg)
+{
+ return &tcp_sockets_allocated;
+}
+EXPORT_SYMBOL(sockets_allocated_tcp);

 void tcp_enter_memory_pressure(struct sock *sk)
 {
@@ -326,6 +335,19 @@ void tcp_enter_memory_pressure(struct sock *sk)
 }
 EXPORT_SYMBOL(tcp_enter_memory_pressure);

+long *tcp_sysctl_mem(const struct mem_cgroup *memcg)
+{
+ return sysctl_tcp_mem;
+}
+EXPORT_SYMBOL(tcp_sysctl_mem);
+
```

```
+long memory_allocated_tcp_add(struct mem_cgroup *memcg, long val,
+         int *parent_status)
+{
+ return atomic_long_add_return(val, &tcp_memory_allocated);
+}
+EXPORT_SYMBOL(memory_allocated_tcp_add);
+
 /* Convert seconds to retransmits based on initial and max timeout */
 static u8 secs_to_retrans(int seconds, int timeout, int rto_max)
 {
diff --git a/net/ipv4/tcp_input.c b/net/ipv4/tcp_input.c
index d73aab3..87520ed 100644
--- a/net/ipv4/tcp_input.c
+++ b/net/ipv4/tcp_input.c
@@ -316,7 +316,7 @@ static void tcp_grow_window(struct sock *sk, struct sk_buff *skb)
  /* Check #1 */
  if (tp->rcv_ssthresh < tp->window_clamp &&
      (int)tp->rcv_ssthresh < tcp_space(sk) &&
-     !tcp_memory_pressure) {
+     !sk_memory_pressure(sk)) {
   int incr;

   /* Check #2. Increase window, if skb with such overhead
@@ -398,8 +398,8 @@ static void tcp_clamp_window(struct sock *sk)

  if (sk->sk_rcvbuf < sysctl_tcp_rmem[2] &&
      !(sk->sk_userlocks & SOCK_RCVBUF_LOCK) &&
-     !tcp_memory_pressure &&
-     atomic_long_read(&tcp_memory_allocated) < sysctl_tcp_mem[0]) {
+     !sk_memory_pressure(sk) &&
+     sk_memory_allocated(sk) < sk_prot_mem(sk, 0)) {
   sk->sk_rcvbuf = min(atomic_read(&sk->sk_rmem_alloc),
         sysctl_tcp_rmem[2]);
  }
@@ -4804,7 +4804,7 @@ static int tcp_prune_queue(struct sock *sk)

  if (atomic_read(&sk->sk_rmem_alloc) >= sk->sk_rcvbuf)
   tcp_clamp_window(sk);
- else if (tcp_memory_pressure)
+ else if (sk_memory_pressure(sk))
   tp->rcv_ssthresh = min(tp->rcv_ssthresh, 4U * tp->advmss);

  tcp_collapse_ofo_queue(sk);
@@ -4870,11 +4870,11 @@ static int tcp_should_expand_sndbuf(struct sock *sk)
   return 0;

  /* If we are under global TCP memory pressure, do not expand.  */
- if (tcp_memory_pressure)
```

```
+ if (sk_memory_pressure(sk))
   return 0;

  /* If we are under soft global TCP memory pressure, do not expand.  */
- if (atomic_long_read(&tcp_memory_allocated) >= sysctl_tcp_mem[0])
+ if (sk_memory_allocated(sk) >= sk_prot_mem(sk, 0))
   return 0;

  /* If we filled the congestion window, do not expand.  */
diff --git a/net/ipv4/tcp_ipv4.c b/net/ipv4/tcp_ipv4.c
index 7963e03..7072060 100644
--- a/net/ipv4/tcp_ipv4.c
+++ b/net/ipv4/tcp_ipv4.c
@@ -1911,7 +1911,7 @@ static int tcp_v4_init_sock(struct sock *sk)
  sk->sk_rcvbuf = sysctl_tcp_rmem[1];

  local_bh_disable();
- percpu_counter_inc(&tcp_sockets_allocated);
+ sk_sockets_allocated_inc(sk);
  local_bh_enable();

  return 0;
@@ -1967,7 +1967,7 @@ void tcp_v4_destroy_sock(struct sock *sk)
   tp->cookie_values = NULL;
  }

- percpu_counter_dec(&tcp_sockets_allocated);
+ sk_sockets_allocated_dec(sk);
 }
 EXPORT_SYMBOL(tcp_v4_destroy_sock);

@@ -2608,11 +2608,11 @@ struct proto tcp_prot = {
  .unhash   = inet_unhash,
  .get_port  = inet_csk_get_port,
  .enter_memory_pressure = tcp_enter_memory_pressure,
- .sockets_allocated = &tcp_sockets_allocated,
+ .memory_pressure = memory_pressure_tcp,
+ .sockets_allocated = sockets_allocated_tcp,
  .orphan_count  = &tcp_orphan_count,
- .memory_allocated = &tcp_memory_allocated,
- .memory_pressure = &tcp_memory_pressure,
- .sysctl_mem  = sysctl_tcp_mem,
+ .mem_allocated_add = memory_allocated_tcp_add,
+ .prot_mem  = tcp_sysctl_mem,
  .sysctl_wmem  = sysctl_tcp_wmem,
  .sysctl_rmem  = sysctl_tcp_rmem,
  .max_header  = MAX_TCP_HEADER,
diff --git a/net/ipv4/tcp_output.c b/net/ipv4/tcp_output.c
```

```
index 882e0b0..06aeb31 100644
--- a/net/ipv4/tcp_output.c
+++ b/net/ipv4/tcp_output.c
@@ -1912,7 +1912,7 @@ u32 __tcp_select_window(struct sock *sk)
  if (free_space < (full_space >> 1)) {
   icsk->icsk_ack.quick = 0;

-  if (tcp_memory_pressure)
+  if (sk_memory_pressure(sk))
    tp->rcv_ssthresh = min(tp->rcv_ssthresh,
           4U * tp->advmss);

diff --git a/net/ipv4/tcp_timer.c b/net/ipv4/tcp_timer.c
index ecd44b0..2c67617 100644
--- a/net/ipv4/tcp_timer.c
+++ b/net/ipv4/tcp_timer.c
@@ -261,7 +261,7 @@ static void tcp_delack_timer(unsigned long data)
  }

 out:
- if (tcp_memory_pressure)
+ if (sk_memory_pressure(sk))
   sk_mem_reclaim(sk);
 out_unlock:
  bh_unlock_sock(sk);
diff --git a/net/ipv4/udp.c b/net/ipv4/udp.c
index 1b5a193..21604b4 100644
--- a/net/ipv4/udp.c
+++ b/net/ipv4/udp.c
@@ -120,9 +120,6 @@ EXPORT_SYMBOL(sysctl_udp_rmem_min);
 int sysctl_udp_wmem_min __read_mostly;
 EXPORT_SYMBOL(sysctl_udp_wmem_min);

-atomic_long_t udp_memory_allocated;
-EXPORT_SYMBOL(udp_memory_allocated);
-
 #define MAX_UDP_PORTS 65536
 #define PORTS_PER_CHAIN (MAX_UDP_PORTS / UDP_HTABLE_SIZE_MIN)

@@ -1918,6 +1915,20 @@ unsigned int udp_poll(struct file *file, struct socket *sock, poll_table
*wait)
 }
 EXPORT_SYMBOL(udp_poll);

+static atomic_long_t udp_memory_allocated;
+long memory_allocated_udp_add(struct mem_cgroup *memcg, long val,
+        int *parent_status)
+{
```

```
+ return atomic_long_add_return(val, &udp_memory_allocated);
+}
+EXPORT_SYMBOL(memory_allocated_udp_add);
+
+long *udp_sysctl_mem(const struct mem_cgroup *memcg)
+{
+ return sysctl_udp_mem;
+}
+EXPORT_SYMBOL(udp_sysctl_mem);
+
 struct proto udp_prot = {
  .name     = "UDP",
  .owner     = THIS_MODULE,
@@ -1936,8 +1947,8 @@ struct proto udp_prot = {
  .unhash     = udp_lib_unhash,
  .rehash     = udp_v4_rehash,
  .get_port    = udp_v4_get_port,
- .memory_allocated  = &udp_memory_allocated,
- .sysctl_mem    = sysctl_udp_mem,
+ .mem_allocated_add = &memory_allocated_udp_add,
+ .prot_mem    = udp_sysctl_mem,
  .sysctl_wmem    = &sysctl_udp_wmem_min,
  .sysctl_rmem    = &sysctl_udp_rmem_min,
  .obj_size    = sizeof(struct udp_sock),
diff --git a/net/ipv6/tcp_ipv6.c b/net/ipv6/tcp_ipv6.c
index 7b8fc57..bdc0003 100644
--- a/net/ipv6/tcp_ipv6.c
+++ b/net/ipv6/tcp_ipv6.c
@@ -1992,7 +1992,7 @@ static int tcp_v6_init_sock(struct sock *sk)
  sk->sk_rcvbuf = sysctl_tcp_rmem[1];

  local_bh_disable();
- percpu_counter_inc(&tcp_sockets_allocated);
+ sk_sockets_allocated_inc(sk);
  local_bh_enable();

  return 0;
@@ -2201,11 +2201,11 @@ struct proto tcpv6_prot = {
  .unhash   = inet_unhash,
  .get_port  = inet_csk_get_port,
  .enter_memory_pressure = tcp_enter_memory_pressure,
- .sockets_allocated = &tcp_sockets_allocated,
- .memory_allocated = &tcp_memory_allocated,
- .memory_pressure = &tcp_memory_pressure,
+ .sockets_allocated = sockets_allocated_tcp,
+ .mem_allocated_add = memory_allocated_tcp_add,
+ .memory_pressure = memory_pressure_tcp,
  .orphan_count  = &tcp_orphan_count,
```

```
- .sysctl_mem  = sysctl_tcp_mem,
+ .prot_mem  = tcp_sysctl_mem,
  .sysctl_wmem  = sysctl_tcp_wmem,
  .sysctl_rmem  = sysctl_tcp_rmem,
  .max_header  = MAX_TCP_HEADER,
diff --git a/net/ipv6/udp.c b/net/ipv6/udp.c
index bb95e8e..0be7cbc 100644
--- a/net/ipv6/udp.c
+++ b/net/ipv6/udp.c
@@ -1465,8 +1465,8 @@ struct proto udpv6_prot = {
  .unhash     = udp_lib_unhash,
  .rehash     = udp_v6_rehash,
  .get_port    = udp_v6_get_port,
- .memory_allocated  = &udp_memory_allocated,
- .sysctl_mem    = sysctl_udp_mem,
+ .mem_allocated_add = memory_allocated_udp_add,
+ .prot_mem    = udp_sysctl_mem,
  .sysctl_wmem    = &sysctl_udp_wmem_min,
  .sysctl_rmem    = &sysctl_udp_rmem_min,
  .obj_size    = sizeof(struct udp6_sock),
diff --git a/net/sctp/socket.c b/net/sctp/socket.c
index 836aa63..8c0cde9 100644
--- a/net/sctp/socket.c
+++ b/net/sctp/socket.c
@@ -119,11 +119,32 @@ static int sctp_memory_pressure;
 static atomic_long_t sctp_memory_allocated;
 struct percpu_counter sctp_sockets_allocated;

+static long *sctp_sysctl_mem(const struct mem_cgroup *memcg)
+{
+ return sysctl_sctp_mem;
+}
+
 static void sctp_enter_memory_pressure(struct sock *sk)
 {
  sctp_memory_pressure = 1;
 }

+static int *memory_pressure_sctp(const struct mem_cgroup *memcg)
+{
+ return &sctp_memory_pressure;
+}
+
+static long memory_allocated_sctp_add(struct mem_cgroup *memcg, long val,
+          int *parent_status)
+{
+ return atomic_long_add_return(val, &sctp_memory_allocated);
+}
```

```
+
+static struct
+percpu_counter *sockets_allocated_sctp(const struct mem_cgroup *memcg)
+{
+ return &sctp_sockets_allocated;
+}

 /* Get the sndbuf space available at the time on the association.  */
 static inline int sctp_wspace(struct sctp_association *asoc)
@@ -6831,13 +6852,13 @@ struct proto sctp_prot = {
  .unhash      = sctp_unhash,
  .get_port    = sctp_get_port,
  .obj_size    =  sizeof(struct sctp_sock),
- .sysctl_mem  =  sysctl_sctp_mem,
+ .prot_mem    =  sctp_sysctl_mem,
  .sysctl_rmem =  sysctl_sctp_rmem,
  .sysctl_wmem =  sysctl_sctp_wmem,
- .memory_pressure = &sctp_memory_pressure,
+ .memory_pressure = memory_pressure_sctp,
  .enter_memory_pressure = sctp_enter_memory_pressure,
- .memory_allocated = &sctp_memory_allocated,
- .sockets_allocated = &sctp_sockets_allocated,
+ .mem_allocated_add = memory_allocated_sctp_add,
+ .sockets_allocated = sockets_allocated_sctp,
 };

 #if defined(CONFIG_IPV6) || defined(CONFIG_IPV6_MODULE)
@@ -6863,12 +6884,12 @@ struct proto sctpv6_prot = {
  .unhash  = sctp_unhash,
  .get_port = sctp_get_port,
  .obj_size = sizeof(struct sctp6_sock),
- .sysctl_mem = sysctl_sctp_mem,
+ .prot_mem = sctp_sysctl_mem,
  .sysctl_rmem = sysctl_sctp_rmem,
  .sysctl_wmem = sysctl_sctp_wmem,
- .memory_pressure = &sctp_memory_pressure,
+ .memory_pressure = memory_pressure_sctp,
  .enter_memory_pressure = sctp_enter_memory_pressure,
- .memory_allocated = &sctp_memory_allocated,
- .sockets_allocated = &sctp_sockets_allocated,
+ .mem_allocated_add = memory_allocated_sctp_add,
+ .sockets_allocated = sockets_allocated_sctp,
 };
 #endif /* defined(CONFIG_IPV6) || defined(CONFIG_IPV6_MODULE) */
--
1.7.6.4
```

Subject: [PATCH v7 4/8] per-cgroup tcp buffers control
Posted by Glauber Costa on Thu, 13 Oct 2011 13:09:38 GMT
View Forum Message <> Reply to Message

With all the infrastructure in place, this patch implements
per-cgroup control for tcp memory pressure handling.

Signed-off-by: Glauber Costa <glommer@parallels.com>
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 | 4 +
 include/net/sock.h | 14 ++++
 include/net/tcp.h | 17 +++++
 mm/memcontrol.c | 141 ++
 net/core/sock.c | 39 ++++++++++-
 net/ipv4/tcp.c | 47 +++++++++-----
 net/ipv4/tcp_ipv4.c | 11 ++++
 net/ipv6/tcp_ipv6.c | 10 +++-
 8 files changed, 255 insertions(+), 28 deletions(-)

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index 99a8ba2..a27dad9 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -393,6 +393,10 @@ void sock_update_memcg(struct sock *sk);
 void sock_release_memcg(struct sock *sk);
 void memcg_sockets_allocated_dec(struct mem_cgroup *memcg, struct proto *prot);
 void memcg_sockets_allocated_inc(struct mem_cgroup *memcg, struct proto *prot);
+int tcp_init_cgroup(const struct proto *prot, struct cgroup *cgrp,
+   struct cgroup_subsys *ss);
+void tcp_destroy_cgroup(const struct proto *prot, struct cgroup *cgrp,
+   struct cgroup_subsys *ss);
 #else
 /* memcontrol includes sockets.h, that includes memcontrol.h ... */
 static inline void memcg_sockets_allocated_dec(struct mem_cgroup *memcg,
diff --git a/include/net/sock.h b/include/net/sock.h
index 163f87b..efd7664 100644
--- a/include/net/sock.h
+++ b/include/net/sock.h
@@ -64,6 +64,8 @@
 #include <net/dst.h>
 #include <net/checksum.h>

+int sockets_populate(struct cgroup *cgrp, struct cgroup_subsys *ss);
+void sockets_destroy(struct cgroup *cgrp, struct cgroup_subsys *ss);
 /*
  * This structure really needs to be cleaned up.

```
  * Most of it is for TCP, and not used by any of
@@ -819,6 +821,18 @@ struct proto {
  /* Pointer to the per-cgroup version of the the sysctl_mem field */
  long   *(*prot_mem)(const struct mem_cgroup *memcg);

+ /*
+  * cgroup specific init/deinit functions. Called once for all
+  * protocols that implement it, from cgroups populate function.
+  * This function has to setup any files the protocol want to
+  * appear in the kmem cgroup filesystem.
+  */
+ int   (*init_cgroup)(const struct proto *prot,
+            struct cgroup *cgrp,
+            struct cgroup_subsys *ss);
+ void   (*destroy_cgroup)(const struct proto *prot,
+        struct cgroup *cgrp,
+        struct cgroup_subsys *ss);
  int   *sysctl_wmem;
  int   *sysctl_rmem;
  int   max_header;
diff --git a/include/net/tcp.h b/include/net/tcp.h
index eac7bf6..ec57cf2 100644
--- a/include/net/tcp.h
+++ b/include/net/tcp.h
@@ -31,6 +31,7 @@
 #include <linux/crypto.h>
 #include <linux/cryptohash.h>
 #include <linux/kref.h>
+#include <linux/res_counter.h>

 #include <net/inet_connection_sock.h>
 #include <net/inet_timewait_sock.h>
@@ -255,6 +256,21 @@ extern int sysctl_tcp_thin_linear_timeouts;
 extern int sysctl_tcp_thin_dupack;

 struct mem_cgroup;
+struct tcp_memcontrol {
+ /* per-cgroup tcp memory pressure knobs */
+ struct res_counter tcp_memory_allocated;
+ struct percpu_counter tcp_sockets_allocated;
+ /* those two are read-mostly, leave them at the end */
+ long tcp_prot_mem[3];
+ int tcp_memory_pressure;
+};
+
+extern long *tcp_sysctl_mem_nocg(const struct mem_cgroup *memcg);
+struct percpu_counter *sockets_allocated_tcp_nocg(const struct mem_cgroup *memcg);
+int *memory_pressure_tcp_nocg(const struct mem_cgroup *memcg);
```

```
+long memory_allocated_tcp_add_nocg(struct mem_cgroup *memcg, long val,
+       int *parent_status);
+
 extern long *tcp_sysctl_mem(const struct mem_cgroup *memcg);
 struct percpu_counter *sockets_allocated_tcp(const struct mem_cgroup *memcg);
 int *memory_pressure_tcp(const struct mem_cgroup *memcg);
@@ -1023,6 +1039,7 @@ static inline void tcp_openreq_init(struct request_sock *req,
  ireq->loc_port = tcp_hdr(skb)->dest;
 }

+extern void tcp_enter_memory_pressure_nocg(struct sock *sk);
 extern void tcp_enter_memory_pressure(struct sock *sk);

 static inline int keepalive_intvl_when(const struct tcp_sock *tp)
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 4e71fd8..4e79171 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -49,6 +49,9 @@
 #include <linux/cpu.h>
 #include <linux/oom.h>
 #include "internal.h"
+#ifdef CONFIG_INET
+#include <net/tcp.h>
+#endif

 #include <asm/uaccess.h>

@@ -294,6 +297,10 @@ struct mem_cgroup {
   */
  struct mem_cgroup_stat_cpu nocpu_base;
  spinlock_t pcp_counter_lock;
+
+#ifdef CONFIG_INET
+ struct tcp_memcontrol tcp;
+#endif
 };

 /* Stuffs for move charges at task migration. */
@@ -377,10 +384,12 @@ enum mem_type {
 #define MEM_CGROUP_RECLAIM_SOFT  (1 << MEM_CGROUP_RECLAIM_SOFT_BIT)

 static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *memcg);
+static struct mem_cgroup *mem_cgroup_from_cont(struct cgroup *cont);
 /* Writing them here to avoid exposing memcg's inner layout */
 #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
 #ifdef CONFIG_INET
 #include <net/sock.h>
```

```
+#include <net/ip.h>

 void sock_update_memcg(struct sock *sk)
 {
@@ -426,6 +435,119 @@ void memcg_sockets_allocated_inc(struct mem_cgroup *memcg,
struct proto *prot)
   percpu_counter_inc(prot->sockets_allocated(memcg));
 }
 EXPORT_SYMBOL(memcg_sockets_allocated_inc);
+
+/*
+ * Pressure flag: try to collapse.
+ * Technical note: it is used by multiple contexts non atomically.
+ * All the __sk_mem_schedule() is of this nature: accounting
+ * is strict, actions are advisory and have some latency.
+ */
+void tcp_enter_memory_pressure(struct sock *sk)
+{
+ struct mem_cgroup *memcg = sk->sk_cgrp;
+ if (!memcg->tcp.tcp_memory_pressure) {
+  NET_INC_STATS(sock_net(sk), LINUX_MIB_TCPMEMORYPRESSURES);
+  memcg->tcp.tcp_memory_pressure = 1;
+ }
+}
+EXPORT_SYMBOL(tcp_enter_memory_pressure);
+
+#define CONSTCG(m) ((struct mem_cgroup *)(m))
+long *tcp_sysctl_mem(const struct mem_cgroup *memcg)
+{
+ return CONSTCG(memcg)->tcp.tcp_prot_mem;
+}
+EXPORT_SYMBOL(tcp_sysctl_mem);
+
+/*
+ * We will be passed a value in pages. But our limits are internally
+ * all in bytes. We need to convert it before testing the allocation,
+ * and convert it back when returning data to the network layer
+ */
+long memory_allocated_tcp_add(struct mem_cgroup *memcg, long val,
+         int *parent_status)
+{
+ int ret = 0;
+ struct res_counter *failed;
+
+ if (val > 0) {
+  val <<= PAGE_SHIFT;
+  ret = res_counter_charge(&memcg->tcp.tcp_memory_allocated,
+       val, &failed);
```

```
+  if (!ret)
+   *parent_status = UNDER_LIMIT;
+  else
+   *parent_status = OVER_LIMIT;
+ } else if (val < 0) {
+  if (*parent_status == OVER_LIMIT)
+   /*
+    * res_counter charge already surely uncharged the
+    * parent if something went wrong.
+    */
+   WARN_ON(1);
+  else {
+   val = (-val) << PAGE_SHIFT;
+   res_counter_uncharge(&memcg->tcp.tcp_memory_allocated,
+          val);
+  }
+ }
+
+ return res_counter_read_u64(&memcg->tcp.tcp_memory_allocated,
+        RES_USAGE) >> PAGE_SHIFT;
+}
+EXPORT_SYMBOL(memory_allocated_tcp_add);
+
+int *memory_pressure_tcp(const struct mem_cgroup *memcg)
+{
+ return &CONSTCG(memcg)->tcp.tcp_memory_pressure;
+}
+EXPORT_SYMBOL(memory_pressure_tcp);
+
+struct percpu_counter *sockets_allocated_tcp(const struct mem_cgroup *memcg)
+{
+ return &CONSTCG(memcg)->tcp.tcp_sockets_allocated;
+}
+EXPORT_SYMBOL(sockets_allocated_tcp);
+
+static void tcp_create_cgroup(struct mem_cgroup *cg, struct cgroup_subsys *ss)
+{
+ struct res_counter *parent_res_counter = NULL;
+ struct mem_cgroup *parent = parent_mem_cgroup(cg);
+
+ if (parent)
+  parent_res_counter = &parent->tcp.tcp_memory_allocated;
+
+ cg->tcp.tcp_memory_pressure = 0;
+ res_counter_init(&cg->tcp.tcp_memory_allocated, parent_res_counter);
+ percpu_counter_init(&cg->tcp.tcp_sockets_allocated, 0);
+}
+
```

```
+int tcp_init_cgroup(const struct proto *prot, struct cgroup *cgrp,
+      struct cgroup_subsys *ss)
+{
+ struct mem_cgroup *memcg = mem_cgroup_from_cont(cgrp);
+ /*
+  * We need to initialize it at populate, not create time.
+  * This is because net sysctl tables are not up until much
+  * later
+  */
+ memcg->tcp.tcp_prot_mem[0] = sysctl_tcp_mem[0];
+ memcg->tcp.tcp_prot_mem[1] = sysctl_tcp_mem[1];
+ memcg->tcp.tcp_prot_mem[2] = sysctl_tcp_mem[2];
+
+ return 0;
+}
+EXPORT_SYMBOL(tcp_init_cgroup);
+
+void tcp_destroy_cgroup(const struct proto *prot, struct cgroup *cgrp,
+   struct cgroup_subsys *ss)
+{
+ struct mem_cgroup *memcg = mem_cgroup_from_cont(cgrp);
+
+ percpu_counter_destroy(&memcg->tcp.tcp_sockets_allocated);
+}
+EXPORT_SYMBOL(tcp_destroy_cgroup);
+#undef CONSTCG
 #endif /* CONFIG_INET */
 #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */

@@ -4833,14 +4955,27 @@ static int register_kmem_files(struct cgroup *cont, struct
cgroup_subsys *ss)

  ret = cgroup_add_files(cont, ss, kmem_cgroup_files,
          ARRAY_SIZE(kmem_cgroup_files));
+
+ if (!ret)
+  ret = sockets_populate(cont, ss);
  return ret;
 };

+static void kmem_cgroup_destroy(struct cgroup_subsys *ss,
+    struct cgroup *cont)
+{
+ sockets_destroy(cont, ss);
+}
 #else
 static int register_kmem_files(struct cgroup *cont, struct cgroup_subsys *ss)
 {
```

```
  return 0;
 }
+
+static void kmem_cgroup_destroy(struct cgroup_subsys *ss,
+    struct cgroup *cont)
+{
+}
 #endif

 static int alloc_mem_cgroup_per_zone_info(struct mem_cgroup *mem, int node)
@@ -5058,6 +5193,10 @@ mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
  mem->last_scanned_node = MAX_NUMNODES;
  INIT_LIST_HEAD(&mem->oom_notify);

+#if defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM) && defined(CONFIG_INET)
+ tcp_create_cgroup(mem, ss);
+#endif
+
  if (parent)
   mem->swappiness = mem_cgroup_swappiness(parent);
  atomic_set(&mem->refcnt, 1);
@@ -5083,6 +5222,8 @@ static void mem_cgroup_destroy(struct cgroup_subsys *ss,
 {
  struct mem_cgroup *mem = mem_cgroup_from_cont(cont);

+ kmem_cgroup_destroy(ss, cont);
+
  mem_cgroup_put(mem);
 }

diff --git a/net/core/sock.c b/net/core/sock.c
index 22ef143..3fa3ccb 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -135,6 +135,42 @@
 #include <net/tcp.h>
 #endif

+static DEFINE_RWLOCK(proto_list_lock);
+static LIST_HEAD(proto_list);
+
+int sockets_populate(struct cgroup *cgrp, struct cgroup_subsys *ss)
+{
+ struct proto *proto;
+ int ret = 0;
+
+ read_lock(&proto_list_lock);
+ list_for_each_entry(proto, &proto_list, node) {
```

```
+  if (proto->init_cgroup)
+   ret = proto->init_cgroup(proto, cgrp, ss);
+   if (ret)
+    goto out;
+ }
+
+ read_unlock(&proto_list_lock);
+ return ret;
+out:
+ list_for_each_entry_continue_reverse(proto, &proto_list, node)
+  if (proto->destroy_cgroup)
+   proto->destroy_cgroup(proto, cgrp, ss);
+ read_unlock(&proto_list_lock);
+ return ret;
+}
+
+void sockets_destroy(struct cgroup *cgrp, struct cgroup_subsys *ss)
+{
+ struct proto *proto;
+ read_lock(&proto_list_lock);
+ list_for_each_entry_reverse(proto, &proto_list, node)
+  if (proto->destroy_cgroup)
+   proto->destroy_cgroup(proto, cgrp, ss);
+ read_unlock(&proto_list_lock);
+}
+
 /*
  * Each address family might have different locking rules, so we have
  * one slock key per address family:
@@ -2262,9 +2298,6 @@ void sk_common_release(struct sock *sk)
 }
 EXPORT_SYMBOL(sk_common_release);

-static DEFINE_RWLOCK(proto_list_lock);
-static LIST_HEAD(proto_list);
-
 #ifdef CONFIG_PROC_FS
 #define PROTO_INUSE_NR 64 /* should be enough for the first time */
 struct prot_inuse {
diff --git a/net/ipv4/tcp.c b/net/ipv4/tcp.c
index dc8f01e..259f6d9 100644
--- a/net/ipv4/tcp.c
+++ b/net/ipv4/tcp.c
@@ -290,13 +290,6 @@ EXPORT_SYMBOL(sysctl_tcp_mem);
 EXPORT_SYMBOL(sysctl_tcp_rmem);
 EXPORT_SYMBOL(sysctl_tcp_wmem);

-atomic_long_t tcp_memory_allocated; /* Current allocated memory. */
```

```
-
-/*
- * Current number of TCP sockets.
- */
-struct percpu_counter tcp_sockets_allocated;
-
 /*
  * TCP splice context
  */
@@ -306,47 +299,51 @@ struct tcp_splice_state {
  unsigned int flags;
 };

-/*
- * Pressure flag: try to collapse.
- * Technical note: it is used by multiple contexts non atomically.
- * All the __sk_mem_schedule() is of this nature: accounting
- * is strict, actions are advisory and have some latency.
- */
+/* Current number of TCP sockets. */
+struct percpu_counter tcp_sockets_allocated;
+atomic_long_t tcp_memory_allocated; /* Current allocated memory. */
 int tcp_memory_pressure __read_mostly;

-int *memory_pressure_tcp(const struct mem_cgroup *memcg)
+int *memory_pressure_tcp_nocg(const struct mem_cgroup *memcg)
 {
  return &tcp_memory_pressure;
 }
-EXPORT_SYMBOL(memory_pressure_tcp);
+EXPORT_SYMBOL(memory_pressure_tcp_nocg);

-struct percpu_counter *sockets_allocated_tcp(const struct mem_cgroup *memcg)
+struct percpu_counter
+*sockets_allocated_tcp_nocg(const struct mem_cgroup *memcg)
 {
  return &tcp_sockets_allocated;
 }
-EXPORT_SYMBOL(sockets_allocated_tcp);
+EXPORT_SYMBOL(sockets_allocated_tcp_nocg);

-void tcp_enter_memory_pressure(struct sock *sk)
+/*
+ * Pressure flag: try to collapse.
+ * Technical note: it is used by multiple contexts non atomically.
+ * All the __sk_mem_schedule() is of this nature: accounting
+ * is strict, actions are advisory and have some latency.
+ */
```

```
+void tcp_enter_memory_pressure_nocg(struct sock *sk)
 {
  if (!tcp_memory_pressure) {
   NET_INC_STATS(sock_net(sk), LINUX_MIB_TCPMEMORYPRESSURES);
   tcp_memory_pressure = 1;
  }
 }
-EXPORT_SYMBOL(tcp_enter_memory_pressure);
+EXPORT_SYMBOL(tcp_enter_memory_pressure_nocg);

-long *tcp_sysctl_mem(const struct mem_cgroup *memcg)
+long *tcp_sysctl_mem_nocg(const struct mem_cgroup *memcg)
 {
  return sysctl_tcp_mem;
 }
-EXPORT_SYMBOL(tcp_sysctl_mem);
+EXPORT_SYMBOL(tcp_sysctl_mem_nocg);

-long memory_allocated_tcp_add(struct mem_cgroup *memcg, long val,
-         int *parent_status)
+long memory_allocated_tcp_add_nocg(struct mem_cgroup *memcg, long val,
+        int *parent_status)
 {
  return atomic_long_add_return(val, &tcp_memory_allocated);
 }
-EXPORT_SYMBOL(memory_allocated_tcp_add);
+EXPORT_SYMBOL(memory_allocated_tcp_add_nocg);

 /* Convert seconds to retransmits based on initial and max timeout */
 static u8 secs_to_retrans(int seconds, int timeout, int rto_max)
@@ -3248,7 +3245,9 @@ void __init tcp_init(void)

  BUILD_BUG_ON(sizeof(struct tcp_skb_cb) > sizeof(skb->cb));

+#ifndef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
  percpu_counter_init(&tcp_sockets_allocated, 0);
+#endif
  percpu_counter_init(&tcp_orphan_count, 0);
  tcp_hashinfo.bind_bucket_cachep =
   kmem_cache_create("tcp_bind_bucket",
diff --git a/net/ipv4/tcp_ipv4.c b/net/ipv4/tcp_ipv4.c
index 7072060..aac71e9 100644
--- a/net/ipv4/tcp_ipv4.c
+++ b/net/ipv4/tcp_ipv4.c
@@ -2607,12 +2607,23 @@ struct proto tcp_prot = {
  .hash   = inet_hash,
  .unhash   = inet_unhash,
  .get_port  = inet_csk_get_port,
```

```
+ .orphan_count  = &tcp_orphan_count,
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ .init_cgroup  = tcp_init_cgroup,
+ .destroy_cgroup  = tcp_destroy_cgroup,
  .enter_memory_pressure = tcp_enter_memory_pressure,
  .memory_pressure = memory_pressure_tcp,
  .sockets_allocated = sockets_allocated_tcp,
  .orphan_count  = &tcp_orphan_count,
  .mem_allocated_add = memory_allocated_tcp_add,
  .prot_mem  = tcp_sysctl_mem,
+#else
+ .enter_memory_pressure = tcp_enter_memory_pressure_nocg,
+ .memory_pressure = memory_pressure_tcp_nocg,
+ .sockets_allocated = sockets_allocated_tcp_nocg,
+ .mem_allocated_add = memory_allocated_tcp_add_nocg,
+ .prot_mem  = tcp_sysctl_mem_nocg,
+#endif
  .sysctl_wmem  = sysctl_tcp_wmem,
  .sysctl_rmem  = sysctl_tcp_rmem,
  .max_header  = MAX_TCP_HEADER,
diff --git a/net/ipv6/tcp_ipv6.c b/net/ipv6/tcp_ipv6.c
index bdc0003..0a52587 100644
--- a/net/ipv6/tcp_ipv6.c
+++ b/net/ipv6/tcp_ipv6.c
@@ -2200,12 +2200,20 @@ struct proto tcpv6_prot = {
  .hash   = tcp_v6_hash,
  .unhash   = inet_unhash,
  .get_port  = inet_csk_get_port,
+ .orphan_count  = &tcp_orphan_count,
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
  .enter_memory_pressure = tcp_enter_memory_pressure,
  .sockets_allocated = sockets_allocated_tcp,
  .mem_allocated_add = memory_allocated_tcp_add,
  .memory_pressure = memory_pressure_tcp,
- .orphan_count  = &tcp_orphan_count,
  .prot_mem  = tcp_sysctl_mem,
+#else
+ .enter_memory_pressure = tcp_enter_memory_pressure_nocg,
+ .sockets_allocated = sockets_allocated_tcp_nocg,
+ .mem_allocated_add = memory_allocated_tcp_add_nocg,
+ .memory_pressure = memory_pressure_tcp_nocg,
+ .prot_mem  = tcp_sysctl_mem_nocg,
+#endif
  .sysctl_wmem  = sysctl_tcp_wmem,
  .sysctl_rmem  = sysctl_tcp_rmem,
  .max_header  = MAX_TCP_HEADER,
--
1.7.6.4
```

Subject: [PATCH v7 5/8] per-netns ipv4 sysctl_tcp_mem
Posted by Glauber Costa on Thu, 13 Oct 2011 13:09:39 GMT
View Forum Message <> Reply to Message

This patch allows each namespace to independently set up
its levels for tcp memory pressure thresholds. This patch
alone does not buy much: we need to make this values
per group of process somehow. This is achieved in the
patches that follows in this patchset.

Signed-off-by: Glauber Costa <glommer@parallels.com>
Reviewed-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
CC: David S. Miller <davem@davemloft.net>
CC: Eric W. Biederman <ebiederm@xmission.com>
---
 include/net/netns/ipv4.h | 1 +
 include/net/tcp.h | 1 -
 mm/memcontrol.c | 8 ++++--
 net/ipv4/sysctl_net_ipv4.c | 51 ++++++++++++++++++++++++++++++++++++------
 net/ipv4/tcp.c | 13 ++++--------
 5 files changed, 53 insertions(+), 21 deletions(-)

diff --git a/include/net/netns/ipv4.h b/include/net/netns/ipv4.h
index d786b4f..bbd023a 100644
--- a/include/net/netns/ipv4.h
+++ b/include/net/netns/ipv4.h
@@ -55,6 +55,7 @@ struct netns_ipv4 {
  int current_rt_cache_rebuild_count;

  unsigned int sysctl_ping_group_range[2];
+ long sysctl_tcp_mem[3];

  atomic_t rt_genid;
  atomic_t dev_addr_genid;
diff --git a/include/net/tcp.h b/include/net/tcp.h
index ec57cf2..3609d87 100644
--- a/include/net/tcp.h
+++ b/include/net/tcp.h
@@ -232,7 +232,6 @@ extern int sysctl_tcp_fack;
 extern int sysctl_tcp_reordering;
 extern int sysctl_tcp_ecn;
 extern int sysctl_tcp_dsack;
-extern long sysctl_tcp_mem[3];
 extern int sysctl_tcp_wmem[3];
 extern int sysctl_tcp_rmem[3];
 extern int sysctl_tcp_app_win;
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 4e79171..f953b32 100644
--- a/mm/memcontrol.c

```
+++ b/mm/memcontrol.c
@@ -390,6 +390,7 @@ static struct mem_cgroup *mem_cgroup_from_cont(struct cgroup *cont);
 #ifdef CONFIG_INET
 #include <net/sock.h>
 #include <net/ip.h>
+#include <linux/nsproxy.h>

 void sock_update_memcg(struct sock *sk)
 {
@@ -526,14 +527,15 @@ int tcp_init_cgroup(const struct proto *prot, struct cgroup *cgrp,
       struct cgroup_subsys *ss)
 {
  struct mem_cgroup *memcg = mem_cgroup_from_cont(cgrp);
+ struct net *net = current->nsproxy->net_ns;
  /*
   * We need to initialize it at populate, not create time.
   * This is because net sysctl tables are not up until much
   * later
   */
- memcg->tcp.tcp_prot_mem[0] = sysctl_tcp_mem[0];
- memcg->tcp.tcp_prot_mem[1] = sysctl_tcp_mem[1];
- memcg->tcp.tcp_prot_mem[2] = sysctl_tcp_mem[2];
+ memcg->tcp.tcp_prot_mem[0] = net->ipv4.sysctl_tcp_mem[0];
+ memcg->tcp.tcp_prot_mem[1] = net->ipv4.sysctl_tcp_mem[1];
+ memcg->tcp.tcp_prot_mem[2] = net->ipv4.sysctl_tcp_mem[2];

  return 0;
 }
diff --git a/net/ipv4/sysctl_net_ipv4.c b/net/ipv4/sysctl_net_ipv4.c
index 69fd720..bbd67ab 100644
--- a/net/ipv4/sysctl_net_ipv4.c
+++ b/net/ipv4/sysctl_net_ipv4.c
@@ -14,6 +14,7 @@
 #include <linux/init.h>
 #include <linux/slab.h>
 #include <linux/nsproxy.h>
+#include <linux/swap.h>
 #include <net/snmp.h>
 #include <net/icmp.h>
 #include <net/ip.h>
@@ -174,6 +175,36 @@ static int proc_allowed_congestion_control(ctl_table *ctl,
  return ret;
 }

+static int ipv4_tcp_mem(ctl_table *ctl, int write,
+       void __user *buffer, size_t *lenp,
+       loff_t *ppos)
+{
```

```
+ int ret;
+ unsigned long vec[3];
+ struct net *net = current->nsproxy->net_ns;
+
+ ctl_table tmp = {
+  .data = &vec,
+  .maxlen = sizeof(vec),
+  .mode = ctl->mode,
+ };
+
+ if (!write) {
+  ctl->data = &net->ipv4.sysctl_tcp_mem;
+  return proc_doulongvec_minmax(ctl, write, buffer, lenp, ppos);
+ }
+
+ ret = proc_doulongvec_minmax(&tmp, write, buffer, lenp, ppos);
+ if (ret)
+  return ret;
+
+ net->ipv4.sysctl_tcp_mem[0] = vec[0];
+ net->ipv4.sysctl_tcp_mem[1] = vec[1];
+ net->ipv4.sysctl_tcp_mem[2] = vec[2];
+
+ return 0;
+}
+
 static struct ctl_table ipv4_table[] = {
  {
   .procname = "tcp_timestamps",
@@ -433,13 +464,6 @@ static struct ctl_table ipv4_table[] = {
   .proc_handler = proc_dointvec
  },
  {
-  .procname = "tcp_mem",
-  .data  = &sysctl_tcp_mem,
-  .maxlen  = sizeof(sysctl_tcp_mem),
-  .mode  = 0644,
-  .proc_handler = proc_doulongvec_minmax
- },
- {
   .procname = "tcp_wmem",
   .data  = &sysctl_tcp_wmem,
   .maxlen  = sizeof(sysctl_tcp_wmem),
@@ -721,6 +745,12 @@ static struct ctl_table ipv4_net_table[] = {
   .mode  = 0644,
   .proc_handler = ipv4_ping_group_range,
  },
+ {
```

```
+  .procname = "tcp_mem",
+  .maxlen  = sizeof(init_net.ipv4.sysctl_tcp_mem),
+  .mode  = 0644,
+  .proc_handler = ipv4_tcp_mem,
+ },
  { }
 };

@@ -734,6 +764,7 @@ EXPORT_SYMBOL_GPL(net_ipv4_ctl_path);
 static __net_init int ipv4_sysctl_init_net(struct net *net)
 {
  struct ctl_table *table;
+ unsigned long limit;

  table = ipv4_net_table;
  if (!net_eq(net, &init_net)) {
@@ -769,6 +800,12 @@ static __net_init int ipv4_sysctl_init_net(struct net *net)

  net->ipv4.sysctl_rt_cache_rebuild_count = 4;

+ limit = nr_free_buffer_pages() / 8;
+ limit = max(limit, 128UL);
+ net->ipv4.sysctl_tcp_mem[0] = limit / 4 * 3;
+ net->ipv4.sysctl_tcp_mem[1] = limit;
+ net->ipv4.sysctl_tcp_mem[2] = net->ipv4.sysctl_tcp_mem[0] * 2;
+
  net->ipv4.ipv4_hdr = register_net_sysctl_table(net,
    net_ipv4_ctl_path, table);
  if (net->ipv4.ipv4_hdr == NULL)
diff --git a/net/ipv4/tcp.c b/net/ipv4/tcp.c
index 259f6d9..b1abebd 100644
--- a/net/ipv4/tcp.c
+++ b/net/ipv4/tcp.c
@@ -282,11 +282,9 @@ int sysctl_tcp_fin_timeout __read_mostly = TCP_FIN_TIMEOUT;
 struct percpu_counter tcp_orphan_count;
 EXPORT_SYMBOL_GPL(tcp_orphan_count);

-long sysctl_tcp_mem[3] __read_mostly;
 int sysctl_tcp_wmem[3] __read_mostly;
 int sysctl_tcp_rmem[3] __read_mostly;

-EXPORT_SYMBOL(sysctl_tcp_mem);
 EXPORT_SYMBOL(sysctl_tcp_rmem);
 EXPORT_SYMBOL(sysctl_tcp_wmem);

@@ -334,7 +332,7 @@ EXPORT_SYMBOL(tcp_enter_memory_pressure_nocg);

 long *tcp_sysctl_mem_nocg(const struct mem_cgroup *memcg)
```

```
 {
- return sysctl_tcp_mem;
+ return init_net.ipv4.sysctl_tcp_mem;
 }
 EXPORT_SYMBOL(tcp_sysctl_mem_nocg);

@@ -3298,14 +3296,9 @@ void __init tcp_init(void)
  sysctl_tcp_max_orphans = cnt / 2;
  sysctl_max_syn_backlog = max(128, cnt / 256);

- limit = nr_free_buffer_pages() / 8;
- limit = max(limit, 128UL);
- sysctl_tcp_mem[0] = limit / 4 * 3;
- sysctl_tcp_mem[1] = limit;
- sysctl_tcp_mem[2] = sysctl_tcp_mem[0] * 2;
-
  /* Set per-socket limits to no more than 1/128 the pressure threshold */
- limit = ((unsigned long)sysctl_tcp_mem[1]) << (PAGE_SHIFT - 7);
+ limit = ((unsigned long)init_net.ipv4.sysctl_tcp_mem[1])
+  << (PAGE_SHIFT - 7);
  max_share = min(4UL*1024*1024, limit);

  sysctl_tcp_wmem[0] = SK_MEM_QUANTUM;
--
1.7.6.4
```

Subject: [PATCH v7 6/8] tcp buffer limitation: per-cgroup limit
Posted by Glauber Costa on Thu, 13 Oct 2011 13:09:40 GMT
View Forum Message <> Reply to Message

This patch uses the "tcp_max_mem" field of the kmem_cgroup to
effectively control the amount of kernel memory pinned by a cgroup.

We have to make sure that none of the memory pressure thresholds
specified in the namespace are bigger than the current cgroup.

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

```
 Documentation/cgroups/memory.txt |    1 +
 include/linux/memcontrol.h       |   10 +++++
 mm/memcontrol.c                  |   79 ++++++++++++++++++++++++++++++++++----
 net/ipv4/sysctl_net_ipv4.c       |   20 ++++++++++
 4 files changed, 102 insertions(+), 8 deletions(-)
```

```
diff --git a/Documentation/cgroups/memory.txt b/Documentation/cgroups/memory.txt
index 0dafd70..e773bd7 100644
--- a/Documentation/cgroups/memory.txt
+++ b/Documentation/cgroups/memory.txt
@@ -78,6 +78,7 @@ Brief summary of control files.

  memory.independent_kmem_limit  # select whether or not kernel memory limits are
       independent of user limits
+ memory.kmem.tcp.limit_in_bytes  # set/show hard limit for tcp buf memory

 1. History

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index a27dad9..e0ccec5 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -397,6 +397,9 @@ int tcp_init_cgroup(const struct proto *prot, struct cgroup *cgrp,
       struct cgroup_subsys *ss);
 void tcp_destroy_cgroup(const struct proto *prot, struct cgroup *cgrp,
    struct cgroup_subsys *ss);
+
+unsigned long long tcp_max_memory(const struct mem_cgroup *memcg);
+void tcp_prot_mem(struct mem_cgroup *memcg, long val, int idx);
 #else
 /* memcontrol includes sockets.h, that includes memcontrol.h ... */
 static inline void memcg_sockets_allocated_dec(struct mem_cgroup *memcg,
@@ -413,6 +416,13 @@ static inline void sock_update_memcg(struct sock *sk)
 static inline void sock_release_memcg(struct sock *sk)
 {
 }
+static inline unsigned long long tcp_max_memory(const struct mem_cgroup *memcg)
+{
+ return -1ULL;
+}
+static inline void tcp_prot_mem(struct mem_cgroup *memcg, long val, int idx)
+{
+}
 #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
 #endif /* CONFIG_INET */
 #endif /* _LINUX_MEMCONTROL_H */
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index f953b32..b696267 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -365,6 +365,7 @@ enum mem_type {
  _MEMSWAP,
  _OOM_TYPE,
  _KMEM,
```

```
+ _KMEM_TCP,
 };

 #define MEMFILE_PRIVATE(x, val) (((x) << 16) | (val))
@@ -385,6 +386,11 @@ enum mem_type {

 static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *memcg);
 static struct mem_cgroup *mem_cgroup_from_cont(struct cgroup *cont);
+static inline bool mem_cgroup_is_root(struct mem_cgroup *memcg)
+{
+ return (memcg == root_mem_cgroup);
+}
+
 /* Writing them here to avoid exposing memcg's inner layout */
 #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
 #ifdef CONFIG_INET
@@ -510,6 +516,35 @@ struct percpu_counter *sockets_allocated_tcp(const struct mem_cgroup
*memcg)
 }
 EXPORT_SYMBOL(sockets_allocated_tcp);

+static void tcp_update_limit(struct mem_cgroup *memcg, u64 val)
+{
+ struct net *net = current->nsproxy->net_ns;
+ int i;
+
+ val >>= PAGE_SHIFT;
+
+ for (i = 0; i < 3; i++)
+  memcg->tcp.tcp_prot_mem[i]  = min_t(long, val,
+         net->ipv4.sysctl_tcp_mem[i]);
+}
+
+static int mem_cgroup_write(struct cgroup *cont, struct cftype *cft,
+       const char *buffer);
+
+static u64 mem_cgroup_read(struct cgroup *cont, struct cftype *cft);
+/*
+ * We need those things internally in pages, so don't reuse
+ * mem_cgroup_{read,write}
+ */
+static struct cftype tcp_files[] = {
+ {
+  .name = "kmem.tcp.limit_in_bytes",
+  .write_string = mem_cgroup_write,
+  .read_u64 = mem_cgroup_read,
+  .private = MEMFILE_PRIVATE(_KMEM_TCP, RES_LIMIT),
+ },
```

```
+};
+
 static void tcp_create_cgroup(struct mem_cgroup *cg, struct cgroup_subsys *ss)
 {
  struct res_counter *parent_res_counter = NULL;
@@ -527,6 +562,7 @@ int tcp_init_cgroup(const struct proto *prot, struct cgroup *cgrp,
       struct cgroup_subsys *ss)
 {
  struct mem_cgroup *memcg = mem_cgroup_from_cont(cgrp);
+ struct mem_cgroup *parent = parent_mem_cgroup(memcg);
  struct net *net = current->nsproxy->net_ns;
  /*
   * We need to initialize it at populate, not create time.
@@ -537,7 +573,20 @@ int tcp_init_cgroup(const struct proto *prot, struct cgroup *cgrp,
  memcg->tcp.tcp_prot_mem[1] = net->ipv4.sysctl_tcp_mem[1];
  memcg->tcp.tcp_prot_mem[2] = net->ipv4.sysctl_tcp_mem[2];

- return 0;
+ /* Let root cgroup unlimited. All others, respect parent's if needed */
+ if (parent && !parent->use_hierarchy) {
+  unsigned long limit;
+  int ret;
+  limit = nr_free_buffer_pages() / 8;
+  limit = max(limit, 128UL);
+  ret = res_counter_set_limit(&memcg->tcp.tcp_memory_allocated,
+         limit * 2);
+  if (ret)
+   return ret;
+ }
+
+ return cgroup_add_files(cgrp, ss, tcp_files,
+    ARRAY_SIZE(tcp_files));
 }
 EXPORT_SYMBOL(tcp_init_cgroup);

@@ -549,7 +598,18 @@ void tcp_destroy_cgroup(const struct proto *prot, struct cgroup *cgrp,
  percpu_counter_destroy(&memcg->tcp.tcp_sockets_allocated);
 }
 EXPORT_SYMBOL(tcp_destroy_cgroup);
+
+unsigned long long tcp_max_memory(const struct mem_cgroup *memcg)
+{
+ return res_counter_read_u64(&CONSTCG(memcg)->tcp.tcp_memory_allocated,
+        RES_LIMIT);
+}
 #undef CONSTCG
+
+void tcp_prot_mem(struct mem_cgroup *memcg, long val, int idx)
```

```
+{
+ memcg->tcp.tcp_prot_mem[idx] = val;
+}
 #endif /* CONFIG_INET */
 #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */

@@ -1048,12 +1108,6 @@ static struct mem_cgroup *mem_cgroup_get_next(struct
mem_cgroup *iter,
 #define for_each_mem_cgroup_all(iter) \
  for_each_mem_cgroup_tree_cond(iter, NULL, true)

-
-static inline bool mem_cgroup_is_root(struct mem_cgroup *mem)
-{
- return (mem == root_mem_cgroup);
-}
-
 void mem_cgroup_count_vm_event(struct mm_struct *mm, enum vm_event_item idx)
 {
  struct mem_cgroup *mem;
@@ -4071,7 +4125,9 @@ static u64 mem_cgroup_read(struct cgroup *cont, struct cftype *cft)
  case _KMEM:
   val = res_counter_read_u64(&mem->kmem, name);
   break;
-
+ case _KMEM_TCP:
+  val = res_counter_read_u64(&mem->tcp.tcp_memory_allocated, name);
+  break;
  default:
   BUG();
   break;
@@ -4104,6 +4160,13 @@ static int mem_cgroup_write(struct cgroup *cont, struct cftype *cft,
    break;
   if (type == _MEM)
    ret = mem_cgroup_resize_limit(memcg, val);
+#if defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM) && defined(CONFIG_INET)
+  else if (type == _KMEM_TCP) {
+   ret = res_counter_set_limit(&memcg->tcp.tcp_memory_allocated,
+           val);
+   tcp_update_limit(memcg, val);
+  }
+#endif
   else
    ret = mem_cgroup_resize_memsw_limit(memcg, val);
   break;
diff --git a/net/ipv4/sysctl_net_ipv4.c b/net/ipv4/sysctl_net_ipv4.c
index bbd67ab..cdc35f6 100644
--- a/net/ipv4/sysctl_net_ipv4.c
```

```
+++ b/net/ipv4/sysctl_net_ipv4.c
@@ -14,6 +14,7 @@
 #include <linux/init.h>
 #include <linux/slab.h>
 #include <linux/nsproxy.h>
+#include <linux/memcontrol.h>
 #include <linux/swap.h>
 #include <net/snmp.h>
 #include <net/icmp.h>
@@ -182,6 +183,10 @@ static int ipv4_tcp_mem(ctl_table *ctl, int write,
  int ret;
  unsigned long vec[3];
  struct net *net = current->nsproxy->net_ns;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ int i;
+ struct mem_cgroup *cg;
+#endif

  ctl_table tmp = {
   .data = &vec,
@@ -198,6 +203,21 @@ static int ipv4_tcp_mem(ctl_table *ctl, int write,
  if (ret)
   return ret;

+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ rcu_read_lock();
+ cg = mem_cgroup_from_task(current);
+ for (i = 0; i < 3; i++)
+  if (vec[i] > tcp_max_memory(cg)) {
+   rcu_read_unlock();
+   return -EINVAL;
+  }
+
+ tcp_prot_mem(cg, vec[0], 0);
+ tcp_prot_mem(cg, vec[1], 1);
+ tcp_prot_mem(cg, vec[2], 2);
+ rcu_read_unlock();
+#endif
+
  net->ipv4.sysctl_tcp_mem[0] = vec[0];
  net->ipv4.sysctl_tcp_mem[1] = vec[1];
  net->ipv4.sysctl_tcp_mem[2] = vec[2];
--
1.7.6.4
```

Subject: [PATCH v7 7/8] Display current tcp memory allocation in kmem cgroup

This patch introduces kmem.tcp_current_memory file, living in the
kmem_cgroup filesystem. It is a simple read-only file that displays the
amount of kernel memory currently consumed by the cgroup.

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

 Documentation/cgroups/memory.txt | 1 +
 mm/memcontrol.c | 5 +++++
 2 files changed, 6 insertions(+), 0 deletions(-)

diff --git a/Documentation/cgroups/memory.txt b/Documentation/cgroups/memory.txt
index e773bd7..b937a99 100644
--- a/Documentation/cgroups/memory.txt
+++ b/Documentation/cgroups/memory.txt
@@ -79,6 +79,7 @@ Brief summary of control files.
  memory.independent_kmem_limit # select whether or not kernel memory limits are
       independent of user limits
  memory.kmem.tcp.limit_in_bytes # set/show hard limit for tcp buf memory
+ memory.kmem.tcp.usage_in_bytes # show current tcp buf memory allocation

 1. History

diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index b696267..1ba318d 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -543,6 +543,11 @@ static struct cftype tcp_files[] = {
   .read_u64 = mem_cgroup_read,
   .private = MEMFILE_PRIVATE(_KMEM_TCP, RES_LIMIT),
  },
+ {
+ .name = "kmem.tcp.usage_in_bytes",
+ .read_u64 = mem_cgroup_read,
+ .private = MEMFILE_PRIVATE(_KMEM_TCP, RES_USAGE),
+ },
 };

 static void tcp_create_cgroup(struct mem_cgroup *cg, struct cgroup_subsys *ss)
--
1.7.6.4

Subject: [PATCH v7 8/8] Disable task moving when using kernel memory accounting
Posted by Glauber Costa on Thu, 13 Oct 2011 13:09:42 GMT
View Forum Message <> Reply to Message

Since this code is still experimental, we are leaving the exact
details of how to move tasks between cgroups when kernel memory
accounting is used as future work.

For now, we simply disallow movement if there are any pending
accounted memory.

Signed-off-by: Glauber Costa <glommer@parallels.com>
Reviewed-by: KAMEZAWA Hiroyuki<kamezawa.hiroyu@jp.fujitsu.com>
---
 mm/memcontrol.c | 31 ++++++++++++++++++----------
 1 files changed, 18 insertions(+), 13 deletions(-)

diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 1ba318d..b46232b 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -408,23 +408,11 @@ void sock_update_memcg(struct sock *sk)

  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
- * the top level won't change under our noses. Incrementing the
- * reference count via cgroup_exclude_rmdir guarantees that.
- */
- cgroup_exclude_rmdir(mem_cgroup_css(sk->sk_cgrp));
  rcu_read_unlock();
 }

 void sock_release_memcg(struct sock *sk)
 {
- cgroup_release_and_wakeup_rmdir(mem_cgroup_css(sk->sk_cgrp));
 }

 void memcg_sockets_allocated_dec(struct mem_cgroup *memcg, struct proto *prot)
@@ -5634,10 +5622,17 @@ static int mem_cgroup_can_attach(struct cgroup_subsys *ss,
 {
  int ret = 0;

```
  struct mem_cgroup *mem = mem_cgroup_from_cont(cgroup);
+ struct mem_cgroup *from = mem_cgroup_from_task(p);
+
+ if (from != mem &&
+     res_counter_read_u64(&mem->tcp.tcp_memory_allocated, RES_USAGE)) {
+  printk(KERN_WARNING "Can't move tasks between cgroups: "
+   "Kernel memory held. task: %s\n", p->comm);
+  return 1;
+ }

  if (mem->move_charge_at_immigrate) {
   struct mm_struct *mm;
-  struct mem_cgroup *from = mem_cgroup_from_task(p);

   VM_BUG_ON(from == mem);

@@ -5805,6 +5800,16 @@ static int mem_cgroup_can_attach(struct cgroup_subsys *ss,
     struct cgroup *cgroup,
     struct task_struct *p)
 {
+ struct mem_cgroup *mem = mem_cgroup_from_cont(cgroup);
+ struct mem_cgroup *from = mem_cgroup_from_task(p);
+
+ if (from != mem &&
+     res_counter_read_u64(&mem->tcp.tcp_memory_allocated, RES_USAGE)) {
+  printk(KERN_WARNING "Can't move tasks between cgroups: "
+   "Kernel memory held. task: %s\n", p->comm);
+  return 1;
+ }
+
  return 0;
 }
 static void mem_cgroup_cancel_attach(struct cgroup_subsys *ss,
--
1.7.6.4
```

Subject: Re: [PATCH v7 0/8] Request for inclusion: tcp memory buffers
Posted by davem on Thu, 13 Oct 2011 20:00:31 GMT

View Forum Message <> Reply to Message

From: Glauber Costa <glommer@parallels.com>
Date: Thu, 13 Oct 2011 17:09:34 +0400

> This series was extensively reviewed over the past month, and after
> all major comments were merged, I feel it is ready for inclusion when
> the next merge window opens. Minor fixes will be provided if they
> prove to be necessary.

I'm not applying this.

You're turning inline increments and decrements of the existing memory
limits into indirect function calls.

That imposes a new non-trivial cost, in fast paths, even when people
do not use your feature.

Make this evaluate into exactly the same exact code stream we have
now when the memory cgroup feature is not in use, which will be the
majority of users.

Subject: Re: [PATCH v7 0/8] Request for inclusion: tcp memory buffers
Posted by Glauber Costa on Thu, 13 Oct 2011 20:05:58 GMT
View Forum Message <> Reply to Message

On 10/14/2011 12:00 AM, David Miller wrote:
> From: Glauber Costa<glommer@parallels.com>
> Date: Thu, 13 Oct 2011 17:09:34 +0400
>
>> This series was extensively reviewed over the past month, and after
>> all major comments were merged, I feel it is ready for inclusion when
>> the next merge window opens. Minor fixes will be provided if they
>> prove to be necessary.
>
> I'm not applying this.

Thank you for letting me now about your view of this that early.

> You're turning inline increments and decrements of the existing memory
> limits into indirect function calls.

Yes, indeed.

> That imposes a new non-trivial cost, in fast paths, even when people
> do not use your feature.
Well, there is a cost, but all past submissions included round trip
benchmarks.
In none of them I could see any significant slowdown.

> Make this evaluate into exactly the same exact code stream we have
> now when the memory cgroup feature is not in use, which will be the
> majority of users.

What exactly do you mean by "not in use" ? Not compiled in or not
actively being exercised ? If you mean the later, I appreciate tips on

how to achieve it.

Also, I kind of dispute the affirmation that !cgroup will encompass
the majority of users, since cgroups is being enabled by default by
most vendors. All systemd based systems use it extensively, for instance.

Subject: Re: [PATCH v7 0/8] Request for inclusion: tcp memory buffers
Posted by davem on Thu, 13 Oct 2011 20:08:54 GMT
View Forum Message <> Reply to Message

From: Glauber Costa <glommer@parallels.com>
Date: Fri, 14 Oct 2011 00:05:58 +0400

> On 10/14/2011 12:00 AM, David Miller wrote:
>> That imposes a new non-trivial cost, in fast paths, even when people
>> do not use your feature.
> Well, there is a cost, but all past submissions included round trip
> benchmarks.
> In none of them I could see any significant slowdown.

Did you try millions of sockets doing all kinds of different accesses?

Did you check the nanosecond latency of operations over loopback so
that the real cost of you change can be isolated and thus measured
properly?

Subject: Re: [PATCH v7 0/8] Request for inclusion: tcp memory buffers
Posted by davem on Thu, 13 Oct 2011 20:12:21 GMT
View Forum Message <> Reply to Message

From: Glauber Costa <glommer@parallels.com>
Date: Fri, 14 Oct 2011 00:05:58 +0400

> Also, I kind of dispute the affirmation that !cgroup will encompass
> the majority of users, since cgroups is being enabled by default by
> most vendors. All systemd based systems use it extensively, for
> instance.

I will definitely advise people against this, since the cost of having
this on by default is absolutely non-trivial.

People keep asking every few releases "where the heck has my performance
gone" and it's because of creeping features like this. This socket
cgroup feature is a prime example of where that kind of stuff comes
from.

I really get irritated when people go "oh, it's just one indirect
function call" and "oh, it's just one more pointer in struct sock"

We work really hard to _remove_ elements from structures and make them
smaller, and to remove expensive operations from the fast paths.

It might take someone weeks if not months to find a way to make a
patch which compensates for the extra overhead your patches are adding.

And I don't think you fully appreciate that.

Subject: Re: [PATCH v7 0/8] Request for inclusion: tcp memory buffers
Posted by Glauber Costa on Thu, 13 Oct 2011 20:14:40 GMT
View Forum Message <> Reply to Message

On 10/14/2011 12:12 AM, David Miller wrote:
> From: Glauber Costa<glommer@parallels.com>
> Date: Fri, 14 Oct 2011 00:05:58 +0400
>
>> Also, I kind of dispute the affirmation that !cgroup will encompass
>> the majority of users, since cgroups is being enabled by default by
>> most vendors. All systemd based systems use it extensively, for
>> instance.
>
> I will definitely advise people against this, since the cost of having
> this on by default is absolutely non-trivial.
>
> People keep asking every few releases "where the heck has my performance
> gone" and it's because of creeping features like this. This socket
> cgroup feature is a prime example of where that kind of stuff comes
> from.
>
> I really get irritated when people go "oh, it's just one indirect
> function call" and "oh, it's just one more pointer in struct sock"
>
> We work really hard to _remove_ elements from structures and make them
> smaller, and to remove expensive operations from the fast paths.
>
> It might take someone weeks if not months to find a way to make a
> patch which compensates for the extra overhead your patches are adding.
>
> And I don't think you fully appreciate that.

Let's focus on this:
Are you happy, or at least willing to accept, an approach that keep
things as they were with cgroups *compiled out*, or were you referring

to not in use == compiled in, but with no users?

Subject: Re: [PATCH v7 0/8] Request for inclusion: tcp memory buffers
Posted by davem on Thu, 13 Oct 2011 20:16:08 GMT
View Forum Message <> Reply to Message

From: Glauber Costa <glommer@parallels.com>
Date: Fri, 14 Oct 2011 00:05:58 +0400

> Thank you for letting me now about your view of this that early.

I depend upon my colleagues to assist me in the large task that is reviewing
the enormous number of networking patches that get submitted.

Unfortunately, none of them got a chance to review this patch set
seriously, since I know most of them (especially Eric Dumazet) would
balk at the overhead you're proposing to add to our stack, just as I
did.

This is the reality of the situation, and I'm sorry to tell you that
snippy retorts when someone does take the time out to review your work
won't help at all.

Subject: Re: [PATCH v7 0/8] Request for inclusion: tcp memory buffers
Posted by davem on Thu, 13 Oct 2011 20:18:19 GMT
View Forum Message <> Reply to Message

From: Glauber Costa <glommer@parallels.com>
Date: Fri, 14 Oct 2011 00:14:40 +0400

> Are you happy, or at least willing to accept, an approach that keep
> things as they were with cgroups *compiled out*, or were you referring
> to not in use == compiled in, but with no users?

To me these are the same exact thing, because %99 of users will be running
a kernel with every feature turned on in the Kconfig.

Subject: Re: [PATCH v7 0/8] Request for inclusion: tcp memory buffers
Posted by Glauber Costa on Thu, 13 Oct 2011 20:23:33 GMT
View Forum Message <> Reply to Message

On 10/14/2011 12:16 AM, David Miller wrote:
> From: Glauber Costa<glommer@parallels.com>
> Date: Fri, 14 Oct 2011 00:05:58 +0400

>
>> Thank you for letting me now about your view of this that early.
>
> I depend upon my colleagues to assist me in the large task that is reviewing
> the enormous number of networking patches that get submitted.
>
> Unfortunately, none of them got a chance to review this patch set
> seriously, since I know most of them (especially Eric Dumazet) would
> balk at the overhead you're proposing to add to our stack, just as I
> did.
>
> This is the reality of the situation, and I'm sorry to tell you that
> snippy retorts when someone does take the time out to review your work
> won't help at all.
I understand that and appreciate your time.
I'll try to come up with something that addresses this problem in the
next submission.

Subject: Re: [PATCH v7 0/8] Request for inclusion: tcp memory buffers
Posted by Andi Kleen on Fri, 14 Oct 2011 02:12:29 GMT
View Forum Message <> Reply to Message

David Miller <davem@davemloft.net> writes:
>
> Make this evaluate into exactly the same exact code stream we have
> now when the memory cgroup feature is not in use, which will be the
> majority of users.

One possible way may be to guard it with static_branch() for
no limit per cgroup set. That should be as near as practically
possible to the original code.

BTW the thing that usually worries me more is the cache line behaviour
when the feature is in use. In the past some of the namespace patches
have created some extremly hot global cache lines, that hurt on larger
systems (for example the Unix socket regression from uid namespaces that
is still not completly fixed). It would be good to double check that all
important state is distributed properly.

-Andi

--
ak@linux.intel.com -- Speaking for myself only

Subject: Re: [PATCH v7 0/8] Request for inclusion: tcp memory buffers

Posted by Valdis.Kletnieks on Fri, 14 Oct 2011 02:55:59 GMT
View Forum Message <> Reply to Message

On Fri, 14 Oct 2011 00:05:58 +0400, Glauber Costa said:
> On 10/14/2011 12:00 AM, David Miller wrote:

> > Make this evaluate into exactly the same exact code stream we have
> > now when the memory cgroup feature is not in use, which will be the
> > majority of users.
>
> What exactly do you mean by "not in use" ? Not compiled in or not
> actively being exercised ? If you mean the later, I appreciate tips on
> how to achieve it.
>
> Also, I kind of dispute the affirmation that !cgroup will encompass
> the majority of users, since cgroups is being enabled by default by
> most vendors. All systemd based systems use it extensively, for instance.

Yes, systemd requires a kernel that includes cgroups. However, systemd does
not require the memory cgroup feature. As a practical matter, if your patch
doesn't generate equivalent code for the "have cgroups, but no memory cgroup"
situation, it's a non-starter.

Subject: Re: [PATCH v7 0/8] Request for inclusion: tcp memory buffers
Posted by Glauber Costa on Fri, 14 Oct 2011 12:56:23 GMT
View Forum Message <> Reply to Message

On 10/14/2011 12:18 AM, David Miller wrote:
> From: Glauber Costa<glommer@parallels.com>
> Date: Fri, 14 Oct 2011 00:14:40 +0400
>
>> Are you happy, or at least willing to accept, an approach that keep
>> things as they were with cgroups *compiled out*, or were you referring
>> to not in use == compiled in, but with no users?
>
> To me these are the same exact thing, because %99 of users will be running
> a kernel with every feature turned on in the Kconfig.

Ok.
Please let me know what do you think of the following patch. It is still
crude, and applies on top of what I had, for simplicity. I can of course
rework all the series for the next submission. The main idea is:

We basically don't care about accounting if all tasks are in the same,
root, cgroup. So I am using static_branch to enable this code path when
the first !root cgroup is created - a stronger statement than just
enabled/disabled by a runtime option. This should cover every single

user that is not *actively* using cgroups (I understand that most
distros will not only compile it in, but also leave it enabled...). When
this code path is disabled, we should be doing the same as before.

If you think this approach is valid, I am not left with the problem of
how to replace the fields when cgroups are enabled. But for that I guess
I can just copy the struct proto (probably only 2, or at most 3 protos
will need it anyway), and make the new sockets run on this new
structure.

Cheers

```
diff --git a/include/net/sock.h b/include/net/sock.h
index efd7664..2270e50 100644
--- a/include/net/sock.h
+++ b/include/net/sock.h
@@ -807,19 +807,35 @@ struct proto {
   * Add a value in pages to the current memory allocation,
   * and return the current value.
   */
- long   (*mem_allocated_add)(struct mem_cgroup *memcg,
-         long val, int *parent_status);
- /* Pointer to the current number of sockets in this cgroup. */
- struct percpu_counter *(*sockets_allocated)(const struct mem_cgroup *memcg);
+ union {
+  long (*mem_allocated_add)(struct mem_cgroup *memcg,
+         long val, int *parent_status);
+  atomic_long_t *memory_allocated;
+ };
+
+ union {
+  /* Pointer to the current number of sockets in this cgroup. */
+  struct percpu_counter *(*sockets_allocated_cg)(const struct mem_cgroup *memcg);
+  struct percpu_counter   *sockets_allocated;
+ };
+
+ union {
   /*
   * Per cgroup pointer to the pressure flag: try to collapse.
   * Technical note: it is used by multiple contexts non atomically.
   * All the __sk_mem_schedule() is of this nature: accounting
   * is strict, actions are advisory and have some latency.
   */
- int   *(*memory_pressure)(const struct mem_cgroup *memcg);
- /* Pointer to the per-cgroup version of the the sysctl_mem field */
- long   *(*prot_mem)(const struct mem_cgroup *memcg);
+  int   *(*memory_pressure_cg)(const struct mem_cgroup *memcg);
+        int                     *memory_pressure;
```

```
+
+ };
+
+ union {
+  /* Pointer to the per-cgroup version of the the sysctl_mem field */
+  long   *(*prot_mem)(const struct mem_cgroup *memcg);
+         long                    *sysctl_mem;
+ };

  /*
   * cgroup specific init/deinit functions. Called once for all
@@ -891,85 +907,174 @@ static inline void sk_refcnt_debug_release(const struct sock *sk)
 #define sk_refcnt_debug_release(sk) do { } while (0)
 #endif /* SOCK_REFCNT_DEBUG */

+extern struct jump_label_key cgroup_crap_enabled;
 #include <linux/memcontrol.h>
 static inline int *sk_memory_pressure(struct sock *sk)
 {
- int *ret = NULL;
- if (sk->sk_prot->memory_pressure)
-  ret = sk->sk_prot->memory_pressure(sk->sk_cgrp);
- return ret;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&cgroup_crap_enabled)) {
+  int *ret = NULL;
+  if (!sk->sk_cgrp)
+   goto nocgroup;
+  if (sk->sk_prot->memory_pressure)
+   ret = sk->sk_prot->memory_pressure_cg(sk->sk_cgrp);
+  return ret;
+ }
+nocgroup:
+#endif
+ return sk->sk_prot->memory_pressure;
 }

 static inline long sk_prot_mem(struct sock *sk, int index)
 {
- long *prot = sk->sk_prot->prot_mem(sk->sk_cgrp);
+ long *prot;
+ prot = sk->sk_prot->prot_mem(sk->sk_cgrp);
  return prot[index];
+
+#if 0
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&cgroup_crap_enabled)) {
+  long *prot;
```

```
+  if (!sk->sk_cgrp)
+   goto nocgroup;
+  prot = sk->sk_prot->prot_mem(sk->sk_cgrp);
+  return prot[index];
+ }
+nocgroup:
+#endif
+ return sk->sk_prot->sysctl_mem[index];
+#endif
 }

 static inline long
 sk_memory_allocated(struct sock *sk)
 {
  struct proto *prot = sk->sk_prot;
- struct mem_cgroup *cg = sk->sk_cgrp;
-
- return prot->mem_allocated_add(cg, 0, NULL);
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&cgroup_crap_enabled)) {
+  struct mem_cgroup *cg = sk->sk_cgrp;
+  if (!cg) /* this handles the case with existing sockets */
+   goto nocgroup;
+  return prot->mem_allocated_add(cg, 0, NULL);
+ }
+nocgroup:
+#endif
+ return atomic_long_read(prot->memory_allocated);
 }

 static inline long
 sk_memory_allocated_add(struct sock *sk, int amt, int *parent_status)
 {
  struct proto *prot = sk->sk_prot;
- struct mem_cgroup *cg = sk->sk_cgrp;

- return prot->mem_allocated_add(cg, amt, parent_status);
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&cgroup_crap_enabled)) {
+  struct mem_cgroup *cg = sk->sk_cgrp;
+  if (!cg)
+   goto nocgroup;
+  return prot->mem_allocated_add(cg, amt, parent_status);
+ }
+nocgroup:
+#endif
+ return atomic_long_add_return(amt, prot->memory_allocated);
 }
```

```
 static inline void
 sk_memory_allocated_sub(struct sock *sk, int amt, int parent_status)
 {
  struct proto *prot = sk->sk_prot;
- struct mem_cgroup *cg = sk->sk_cgrp;

- prot->mem_allocated_add(cg, -amt, &parent_status);
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&cgroup_crap_enabled)) {
+  struct mem_cgroup *cg = sk->sk_cgrp;
+
+  if (!cg)
+   goto nocgroup;
+
+  prot->mem_allocated_add(cg, -amt, &parent_status);
+ } else
+nocgroup:
+#endif
+ atomic_long_sub(amt, prot->memory_allocated);
 }

 static inline void sk_sockets_allocated_dec(struct sock *sk)
 {
  struct proto *prot = sk->sk_prot;
- struct mem_cgroup *cg = sk->sk_cgrp;

- percpu_counter_dec(prot->sockets_allocated(cg));
- memcg_sockets_allocated_dec(cg, prot);
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&cgroup_crap_enabled)) {
+  struct mem_cgroup *cg = sk->sk_cgrp;
+  if (!cg)
+   goto nocgroup;
+  percpu_counter_dec(prot->sockets_allocated_cg(cg));
+  memcg_sockets_allocated_dec(cg, prot);
+ } else
+nocgroup:
+#endif
+ percpu_counter_dec(prot->sockets_allocated);
 }

 static inline void sk_sockets_allocated_inc(struct sock *sk)
 {
  struct proto *prot = sk->sk_prot;
- struct mem_cgroup *cg = sk->sk_cgrp;
-
- percpu_counter_inc(prot->sockets_allocated(cg));
```

```
- memcg_sockets_allocated_inc(cg, prot);
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&cgroup_crap_enabled)) {
+  struct mem_cgroup *cg = sk->sk_cgrp;
+  if (!cg)
+   goto nocgroup;
+  percpu_counter_inc(prot->sockets_allocated_cg(cg));
+  memcg_sockets_allocated_inc(cg, prot);
+ } else
+nocgroup:
+#endif
+ percpu_counter_inc(prot->sockets_allocated);
 }

 static inline int
 sk_sockets_allocated_read_positive(struct sock *sk)
 {
  struct proto *prot = sk->sk_prot;
- struct mem_cgroup *cg = sk->sk_cgrp;
-
- return percpu_counter_sum_positive(prot->sockets_allocated(cg));
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&cgroup_crap_enabled)) {
+  struct mem_cgroup *cg = sk->sk_cgrp;
+  if (!cg)
+   goto nocgroup;
+  return percpu_counter_sum_positive(prot->sockets_allocated_cg(cg));
+ } else
+nocgroup:
+#endif
+ return percpu_counter_sum_positive(prot->sockets_allocated);
 }

 static inline int
 kcg_sockets_allocated_sum_positive(struct proto *prot, struct mem_cgroup *cg)
 {
- return percpu_counter_sum_positive(prot->sockets_allocated(cg));
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&cgroup_crap_enabled)) {
+  if (!cg)
+   goto nocgroup;
+  return percpu_counter_sum_positive(prot->sockets_allocated_cg(cg));
+ } else
+nocgroup:
+#endif
+ return percpu_counter_sum_positive(prot->sockets_allocated);
 }
```

```
 static inline long
 kcg_memory_allocated(struct proto *prot, struct mem_cgroup *cg)
 {
- return prot->mem_allocated_add(cg, 0, NULL);
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&cgroup_crap_enabled)) {
+  if (!cg)
+   goto nocgroup;
+  return prot->mem_allocated_add(cg, 0, NULL);
+ }
+nocgroup:
+#endif
+ return atomic_long_read(prot->memory_allocated);
 }


diff --git a/include/net/tcp.h b/include/net/tcp.h
index 74296f4..76301a7 100644
--- a/include/net/tcp.h
+++ b/include/net/tcp.h
@@ -275,6 +275,10 @@ int *memory_pressure_tcp(const struct mem_cgroup *memcg);
 long memory_allocated_tcp_add(struct mem_cgroup *memcg, long val,
         int *parent_status);

+extern atomic_long_t tcp_memory_allocated;
+extern int tcp_memory_pressure __read_mostly;
+extern struct percpu_counter tcp_sockets_allocated;
+
 /*
  * The next routines deal with comparing 32 bit unsigned ints
  * and worry about wraparound (automatic with unsigned arithmetic).
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index febf50a..8b67528 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -400,15 +399,22 @@ static inline bool mem_cgroup_is_root(struct mem_cgroup *memcg)

 void sock_update_memcg(struct sock *sk)
 {
+ struct mem_cgroup *memcg;
+
  /* 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);
- rcu_read_unlock();
+
+ if (static_branch(&cgroup_crap_enabled)) {
+  rcu_read_lock();
+  memcg = mem_cgroup_from_task(current);
+  if (!mem_cgroup_is_root(memcg))
+   sk->sk_cgrp = memcg;
+  rcu_read_unlock();
+ }
 }

 void sock_release_memcg(struct sock *sk)
@@ -419,7 +425,7 @@ void memcg_sockets_allocated_dec(struct mem_cgroup *memcg, struct
proto *prot)
 {
  memcg = parent_mem_cgroup(memcg);
  for (; memcg; memcg = parent_mem_cgroup(memcg))
-  percpu_counter_dec(prot->sockets_allocated(memcg));
+  percpu_counter_dec(prot->sockets_allocated_cg(memcg));
 }
 EXPORT_SYMBOL(memcg_sockets_allocated_dec);

@@ -427,7 +433,7 @@ void memcg_sockets_allocated_inc(struct mem_cgroup *memcg, struct
proto *prot)
 {
  memcg = parent_mem_cgroup(memcg);
  for (; memcg; memcg = parent_mem_cgroup(memcg))
-  percpu_counter_inc(prot->sockets_allocated(memcg));
+  percpu_counter_inc(prot->sockets_allocated_cg(memcg));
 }
 EXPORT_SYMBOL(memcg_sockets_allocated_inc);

@@ -5024,19 +5030,28 @@ static struct cftype kmem_cgroup_files[] = {
 static int register_kmem_files(struct cgroup *cont, struct cgroup_subsys *ss)
 {
  int ret = 0;
+ struct mem_cgroup *memcg = mem_cgroup_from_cont(cont);

  ret = cgroup_add_files(cont, ss, kmem_cgroup_files,
         ARRAY_SIZE(kmem_cgroup_files));

  if (!ret)
   ret = sockets_populate(cont, ss);
+
+ if (!mem_cgroup_is_root(memcg))
```

```
+  jump_label_inc(&cgroup_crap_enabled);
+
  return ret;
 };

 static void kmem_cgroup_destroy(struct cgroup_subsys *ss,
     struct cgroup *cont)
 {
+ struct mem_cgroup *memcg = mem_cgroup_from_cont(cont);
  sockets_destroy(cont, ss);
+
+ if (!mem_cgroup_is_root(memcg))
+  jump_label_dec(&cgroup_crap_enabled);
 }
 #else
 static int register_kmem_files(struct cgroup *cont, struct cgroup_subsys *ss)
diff --git a/net/core/sock.c b/net/core/sock.c
index 3fa3ccb..444a527 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -111,6 +111,7 @@
 #include <linux/init.h>
 #include <linux/highmem.h>
 #include <linux/user_namespace.h>
+#include <linux/jump_label.h>

 #include <asm/uaccess.h>
 #include <asm/system.h>
@@ -178,6 +179,9 @@ void sockets_destroy(struct cgroup *cgrp, struct cgroup_subsys *ss)
 static struct lock_class_key af_family_keys[AF_MAX];
 static struct lock_class_key af_family_slock_keys[AF_MAX];

+struct jump_label_key cgroup_crap_enabled;
+EXPORT_SYMBOL(cgroup_crap_enabled);
+
 /*
  * Make lock validator output more readable. (we pre-construct these
  * strings build-time, so that runtime initialization of socket
@@ -2525,8 +2529,11 @@ static void proto_seq_printf(struct seq_file *seq, struct proto *proto)
  struct mem_cgroup *cg = mem_cgroup_from_task(current);
  int *memory_pressure = NULL;

+#if 0
+ /* not important right now */
  if (proto->memory_pressure)
   memory_pressure = proto->memory_pressure(cg);
+#endif
```

```
  seq_printf(seq, "%-9s %4u %6d  %6ld   %-3s %6u   %-3s  %-10s "
    "%2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c
%2c\n",
diff --git a/net/ipv4/tcp.c b/net/ipv4/tcp.c
index b1abebd..c71cfa5 100644
--- a/net/ipv4/tcp.c
+++ b/net/ipv4/tcp.c
@@ -299,8 +299,11 @@ struct tcp_splice_state {

 /* Current number of TCP sockets. */
 struct percpu_counter tcp_sockets_allocated;
+EXPORT_SYMBOL(tcp_sockets_allocated);
 atomic_long_t tcp_memory_allocated; /* Current allocated memory. */
+EXPORT_SYMBOL(tcp_memory_allocated);
 int tcp_memory_pressure __read_mostly;
+EXPORT_SYMBOL(tcp_memory_pressure);

 int *memory_pressure_tcp_nocg(const struct mem_cgroup *memcg)
 {
diff --git a/net/ipv4/tcp_ipv4.c b/net/ipv4/tcp_ipv4.c
index 378a31c..c9724a8 100644
--- a/net/ipv4/tcp_ipv4.c
+++ b/net/ipv4/tcp_ipv4.c
@@ -2608,19 +2608,15 @@ struct proto tcp_prot = {
 #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
  .init_cgroup  = tcp_init_cgroup,
  .destroy_cgroup  = tcp_destroy_cgroup,
- .enter_memory_pressure = tcp_enter_memory_pressure,
- .memory_pressure = memory_pressure_tcp,
- .sockets_allocated = sockets_allocated_tcp,
- .orphan_count  = &tcp_orphan_count,
- .mem_allocated_add = memory_allocated_tcp_add,
- .prot_mem  = tcp_sysctl_mem,
+ .prot_mem  = tcp_sysctl_mem_nocg,
 #else
 - .enter_memory_pressure = tcp_enter_memory_pressure_nocg,
 - .memory_pressure = memory_pressure_tcp_nocg,
 - .sockets_allocated = sockets_allocated_tcp_nocg,
 - .mem_allocated_nocg = memory_allocated_tcp_nocg,
  .prot_mem  = tcp_sysctl_mem_nocg,
 #endif
+ .enter_memory_pressure = tcp_enter_memory_pressure,
+ .memory_pressure = &tcp_memory_pressure,
+ .sockets_allocated = &tcp_sockets_allocated,
+ .orphan_count  = &tcp_orphan_count,
+ .memory_allocated = &tcp_memory_allocated,
  .sysctl_wmem  = sysctl_tcp_wmem,
  .sysctl_rmem  = sysctl_tcp_rmem,
```

```
  .max_header  = MAX_TCP_HEADER,
diff --git a/net/ipv4/udp.c b/net/ipv4/udp.c
index 21604b4..5fa9cf3 100644
--- a/net/ipv4/udp.c
+++ b/net/ipv4/udp.c
@@ -1947,7 +1947,7 @@ struct proto udp_prot = {
  .unhash     = udp_lib_unhash,
  .rehash     = udp_v4_rehash,
  .get_port    = udp_v4_get_port,
- .mem_allocated_add = &memory_allocated_udp_add,
+ .memory_allocated  = &udp_memory_allocated,
  .prot_mem    = udp_sysctl_mem,
  .sysctl_wmem    = &sysctl_udp_wmem_min,
  .sysctl_rmem    = &sysctl_udp_rmem_min,
diff --git a/net/ipv6/tcp_ipv6.c b/net/ipv6/tcp_ipv6.c
index f7f9bc2..5a976fa 100644
--- a/net/ipv6/tcp_ipv6.c
+++ b/net/ipv6/tcp_ipv6.c
@@ -2197,18 +2197,15 @@ struct proto tcpv6_prot = {
  .get_port  = inet_csk_get_port,
  .orphan_count  = &tcp_orphan_count,
 #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
- .enter_memory_pressure = tcp_enter_memory_pressure,
- .sockets_allocated = sockets_allocated_tcp,
- .mem_allocated_add = memory_allocated_tcp_add,
- .memory_pressure = memory_pressure_tcp,
  .prot_mem  = tcp_sysctl_mem,
 #else
- .enter_memory_pressure = tcp_enter_memory_pressure_nocg,
- .sockets_allocated = sockets_allocated_tcp_nocg,
- .mem_allocated_add = memory_allocated_tcp_nocg,
- .memory_pressure = memory_pressure_tcp_nocg,
  .prot_mem  = tcp_sysctl_mem_nocg,
 #endif
+ .enter_memory_pressure = tcp_enter_memory_pressure_nocg,
+ .sockets_allocated = &tcp_sockets_allocated,
+ .memory_allocated = &tcp_memory_allocated,
+ .memory_pressure = &tcp_memory_pressure,
+
  .sysctl_wmem  = sysctl_tcp_wmem,
  .sysctl_rmem  = sysctl_tcp_rmem,
  .max_header  = MAX_TCP_HEADER,
```

File Attachments

1) test.patch, downloaded 809 times

Subject: Re: [PATCH v7 0/8] Request for inclusion: tcp memory buffers
Posted by davem on Wed, 19 Oct 2011 21:09:16 GMT
View Forum Message <> Reply to Message

From: Glauber Costa <glommer@parallels.com>
Date: Fri, 14 Oct 2011 16:56:23 +0400

> Please let me know what do you think of the following patch. It is
> still crude, and applies on top of what I had, for simplicity. I can
> of course rework all the series for the next submission. The main idea
> is:

Thanks, I promise to look at this soon.