Subject: [RFC][PATCH v2 0/7] Scaling msgmni to the amount of lowmem
Posted by Nadia Derbey on Thu, 31 Jan 2008 13:40:18 GMT
View Forum Message <> Reply to Message

Resending the set of patches after Yasunori's remark about having a single
callback on the hotplug memory notifier chain for the ipc subsystem.
(see thread at http://lkml.org/lkml/2008/1/14/196).

Cc'ing linux-mm since I'm adding a notifier block to the memory hotplug
notifier chain.
Also, Cc'ing containers since I'm introducing a change in the ipc namespace
structure.

On large systems we'd like to allow a larger number of message queues.
In some cases up to 32K. However simply setting MSGMNI to a larger value may
cause problems for smaller systems.

The first patch of this series introduces a default maximum number of message
queue ids that scales with the amount of lowmem.

Since msgmni is per namespace and there is no amount of memory dedicated to
each namespace so far, the second patch of this series scales msgmni to
the number of ipc namespaces too.

Since msgmni depends on the amount of memory, it becomes necessary to
recompute it upon memory add/remove.
In the 4th patch, memory hotplug management is added: a notifier block is
registered into the memory hotplug notifier chain for the ipc subsystem.
Since the ipc namespaces are not linked together, they have their own
notification chain: one notifier_block is defined per ipc namespace.
Each time an ipc namespace is created (removed) it registers (unregisters)
its notifier block in (from) the ipcns chain.
The callback routine registered in the memory chain invokes the ipcns
notifier chain with the IPCNS_MEMCHANGE event.
Each callback routine registered in the ipcns namespace, in turn, recomputes
msgmni for the owning namespace.

The 5th patch makes it possible to keep the memory hotplug notifier chain's
lock for a lesser amount of time: instead of directly notifying the ipcns
notifier chain upon memory add/remove, a work item is added to the global
workqueue. When activated, this work item is the one who notifies the ipcns
notifier chain.

Since msgmni depends on the number of ipc namespaces, it becomes necessary
to recompute it upon ipc namespace creation / removal.
The 6th patch uses the ipc namespace notifier chain for that purpose: that

chain is notified each time an ipc namespace is created or removed. This makes
it possible to recompute msgmni for all the namespaces each time one of them
is created or removed.

When msgmni is explicitely set from userspace, we should avoid recomputing it
upon memory add/remove or ipcns creation/removal.
This is what the 7th patch does: it simply unregisters the ipcns callback
routine as soon as msgmni has been changed from procfs or sysctl().

These patches should be applied to 2.6.24, in the following order:

[PATCH 1/7]: ipc_scale_msgmni_with_lowmem.patch
[PATCH 2/7]: ipc_scale_msgmni_with_namespaces.patch
[PATCH 3/7]: ipc_slab_memory_callback_prio_to_const.patch
[PATCH 4/7]: ipc_recompute_msgmni_on_memory_hotplug.patch
[PATCH 5/7]: ipc_ipcns_notification_workqueue.patch
[PATCH 6/7]: ipc_recompute_msgmni_on_ipcns_create_remove.patch
[PATCH 7/7]: ipc_unregister_callback_on_msgmni_manual_change.patch

Andrew, do you think this patchset could be considered for inclusion in -mm?

Regards,
Nadia

--

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

---

Subject: [RFC][PATCH v2 3/7] Defining the slab_memory_callback priority as a constant

Posted by Nadia Derbey on Thu, 31 Jan 2008 13:40:21 GMT
View Forum Message <> Reply to Message

[PATCH 03/07]

This is a trivial patch that defines the priority of slab_memory_callback in
the callback chain as a constant.
This is to prepare for next patch in the series.

Signed-off-by: Nadia Derbey <Nadia.Derbey@bull.net>

```
 include/linux/memory.h |    6 ++++++
 mm/slub.c              |    2 +-
 2 files changed, 7 insertions(+), 1 deletion(-)

Index: linux-2.6.24/include/linux/memory.h
===================================================================
--- linux-2.6.24.orig/include/linux/memory.h 2008-01-29 16:54:38.000000000 +0100
+++ linux-2.6.24/include/linux/memory.h 2008-01-31 10:30:37.000000000 +0100
@@ -54,6 +54,12 @@ struct memory_notify {
 struct notifier_block;
 struct mem_section;

+/*
+ * Priorities for the hotplug memory callback routines (stored in decreasing
+ * order in the callback chain)
+ */
+#define SLAB_CALLBACK_PRI       1
+
 #ifndef CONFIG_MEMORY_HOTPLUG_SPARSE
 static inline int memory_dev_init(void)
 {
Index: linux-2.6.24/mm/slub.c
===================================================================
--- linux-2.6.24.orig/mm/slub.c 2008-01-29 16:54:49.000000000 +0100
+++ linux-2.6.24/mm/slub.c 2008-01-31 10:31:34.000000000 +0100
@@ -2816,7 +2816,7 @@ void __init kmem_cache_init(void)
   kmalloc_caches[0].refcount = -1;
   caches++;

- hotplug_memory_notifier(slab_memory_callback, 1);
+ hotplug_memory_notifier(slab_memory_callback, SLAB_CALLBACK_PRI);
 #endif

   /* Able to allocate the per node structures */

--
```

Subject: [RFC][PATCH v2 4/7] Recomputing msgmni on memory add / remove
Posted by Nadia Derbey on Thu, 31 Jan 2008 13:40:22 GMT
View Forum Message <> Reply to Message

[PATCH 04/07]

This patch introduces the registration of a callback routine that recomputes
msg_ctlmni upon memory add / remove.

A single notifier block is registered in the hotplug memory chain for all the
ipc namespaces.

Since the ipc namespaces are not linked together, they have their own
notification chain: one notifier_block is defined per ipc namespace.

Each time an ipc namespace is created (removed) it registers (unregisters)
its notifier block in (from) the ipcns chain.
The callback routine registered in the memory chain invokes the ipcns
notifier chain with the IPCNS_LOWMEM event.
Each callback routine registered in the ipcns namespace, in turn, recomputes
msgmni for the owning namespace.

Signed-off-by: Nadia Derbey <Nadia.Derbey@bull.net>

---
 include/linux/ipc.h | 38 ++++++++++++++++++++++++++++++++++++
 include/linux/memory.h | 1
 ipc/Makefile | 3 +-
 ipc/ipcns_notifier.c | 69 ++
 ipc/msg.c | 2 -
 ipc/util.c | 44 ++
 ipc/util.h | 2 +
 7 files changed, 157 insertions(+), 2 deletions(-)

Index: linux-2.6.24/include/linux/ipc.h
===
--- linux-2.6.24.orig/include/linux/ipc.h 2008-01-31 10:03:47.000000000 +0100
+++ linux-2.6.24/include/linux/ipc.h 2008-01-31 10:49:07.000000000 +0100
@@ -81,9 +81,20 @@ struct ipc_kludge {

 #include <linux/kref.h>
 #include <linux/spinlock.h>
+#ifdef CONFIG_MEMORY_HOTPLUG
+#include <linux/notifier.h>
+#endif /* CONFIG_MEMORY_HOTPLUG */

 #define IPCMNI 32768 /* <= MAX_INT limit for ipc arrays (including sysctl changes) */

+
+/*
+ * ipc namespace events
+ */
+#define IPCNS_MEMCHANGED   0x00000001   /* Notify lowmem size changed */

```
+
+#define IPCNS_CALLBACK_PRI 0
+
 /* used by in-kernel data structures */
 struct kern_ipc_perm
 {
@@ -118,6 +129,10 @@ struct ipc_namespace {
  size_t  shm_ctlall;
  int  shm_ctlmni;
  int  shm_tot;
+
+#ifdef CONFIG_MEMORY_HOTPLUG
+ struct notifier_block ipcns_nb;
+#endif
 };

 extern struct ipc_namespace init_ipc_ns;
@@ -128,6 +143,29 @@ extern atomic_t nr_ipc_ns;
 extern void free_ipc_ns(struct kref *kref);
 extern struct ipc_namespace *copy_ipcs(unsigned long flags,
       struct ipc_namespace *ns);
+#ifdef CONFIG_MEMORY_HOTPLUG
+
+extern int register_ipcns_notifier(struct ipc_namespace *);
+extern int unregister_ipcns_notifier(struct ipc_namespace *);
+extern int ipcns_notify(unsigned long);
+
+#else /* CONFIG_MEMORY_HOTPLUG */
+
+static inline int register_ipcns_notifier(struct ipc_namespace *ipcns)
+{
+ return 0;
+}
+static inline int unregister_ipcns_notifier(struct ipc_namespace *ipcns)
+{
+ return 0;
+}
+static inline int ipcns_notify(unsigned long ev)
+{
+ return 0;
+}
+
+#endif /* CONFIG_MEMORY_HOTPLUG */
+
 #else
 #define INIT_IPC_NS(ns)
 static inline struct ipc_namespace *copy_ipcs(unsigned long flags,
Index: linux-2.6.24/include/linux/memory.h
```

```
===================================================================
--- linux-2.6.24.orig/include/linux/memory.h 2008-01-31 10:30:37.000000000 +0100
+++ linux-2.6.24/include/linux/memory.h 2008-01-31 10:52:01.000000000 +0100
@@ -59,6 +59,7 @@ struct mem_section;
  * order in the callback chain)
  */
 #define SLAB_CALLBACK_PRI       1
+#define IPC_CALLBACK_PRI        10

 #ifndef CONFIG_MEMORY_HOTPLUG_SPARSE
 static inline int memory_dev_init(void)
Index: linux-2.6.24/ipc/ipcns_notifier.c
===================================================================
--- /dev/null 1970-01-01 00:00:00.000000000 +0000
+++ linux-2.6.24/ipc/ipcns_notifier.c 2008-01-31 10:58:11.000000000 +0100
@@ -0,0 +1,69 @@
+/*
+ * linux/ipc/ipcns_notifier.c
+ * Copyright (C) 2007 BULL SA. Nadia Derbey
+ *
+ * Notification mechanism for ipc namespaces:
+ * The callback routine registered in the memory chain invokes the ipcns
+ * notifier chain with the IPCNS_MEMCHANGED event.
+ * Each callback routine registered in the ipcns namespace recomputes msgmni
+ * for the owning namespace.
+ */
+
+#include <linux/msg.h>
+#include <linux/rcupdate.h>
+#include <linux/notifier.h>
+
+#include "util.h"
+
+
+
+static BLOCKING_NOTIFIER_HEAD(ipcns_chain);
+
+
+static int ipcns_callback(struct notifier_block *self,
+    unsigned long action, void *arg)
+{
+ struct ipc_namespace *ns;
+
+ switch (action) {
+ case IPCNS_MEMCHANGED:   /* amount of lowmem has changed */
+  /*
+   * It's time to recompute msgmni
+   */
```

```
+  ns = container_of(self, struct ipc_namespace, ipcns_nb);
+  /*
+   * No need to get a reference on the ns: the 1st job of
+   * free_ipc_ns() is to unregister the callback routine.
+   * blocking_notifier_chain_unregister takes the wr lock to do
+   * it.
+   * When this callback routine is called the rd lock is held by
+   * blocking_notifier_call_chain.
+   * So the ipc ns cannot be freed while we are here.
+   */
+  recompute_msgmni(ns);
+  break;
+ default:
+  break;
+ }
+
+ return NOTIFY_OK;
+}
+
+int register_ipcns_notifier(struct ipc_namespace *ns)
+{
+ memset(&ns->ipcns_nb, 0, sizeof(ns->ipcns_nb));
+ ns->ipcns_nb.notifier_call = ipcns_callback;
+ ns->ipcns_nb.priority = IPCNS_CALLBACK_PRI;
+ return blocking_notifier_chain_register(&ipcns_chain, &ns->ipcns_nb);
+}
+
+int unregister_ipcns_notifier(struct ipc_namespace *ns)
+{
+ return blocking_notifier_chain_unregister(&ipcns_chain,
+       &ns->ipcns_nb);
+}
+
+int ipcns_notify(unsigned long val)
+{
+ return blocking_notifier_call_chain(&ipcns_chain, val, NULL);
+}
Index: linux-2.6.24/ipc/Makefile
===================================================================
--- linux-2.6.24.orig/ipc/Makefile 2008-01-29 16:55:04.000000000 +0100
+++ linux-2.6.24/ipc/Makefile 2008-01-31 10:59:39.000000000 +0100
@@ -3,7 +3,8 @@
 #

 obj-$(CONFIG_SYSVIPC_COMPAT) += compat.o
-obj-$(CONFIG_SYSVIPC) += util.o msgutil.o msg.o sem.o shm.o
+obj_mem-$(CONFIG_MEMORY_HOTPLUG) += ipcns_notifier.o
+obj-$(CONFIG_SYSVIPC) += util.o msgutil.o msg.o sem.o shm.o $(obj_mem-y)
```

```
 obj-$(CONFIG_SYSVIPC_SYSCTL) += ipc_sysctl.o
 obj_mq-$(CONFIG_COMPAT) += compat_mq.o
 obj-$(CONFIG_POSIX_MQUEUE) += mqueue.o msgutil.o $(obj_mq-y)
Index: linux-2.6.24/ipc/util.c
===================================================================
--- linux-2.6.24.orig/ipc/util.c 2008-01-31 10:05:58.000000000 +0100
+++ linux-2.6.24/ipc/util.c 2008-01-31 11:04:51.000000000 +0100
@@ -33,6 +33,7 @@
 #include <linux/audit.h>
 #include <linux/nsproxy.h>
 #include <linux/rwsem.h>
+#include <linux/memory.h>

 #include <asm/unistd.h>

@@ -54,6 +55,33 @@ struct ipc_namespace init_ipc_ns = {
 atomic_t nr_ipc_ns = ATOMIC_INIT(1);


+#ifdef CONFIG_MEMORY_HOTPLUG
+
+static int ipc_memory_callback(struct notifier_block *self,
+    unsigned long action, void *arg)
+{
+ switch (action) {
+ case MEM_ONLINE:    /* memory successfully brought online */
+ case MEM_OFFLINE:   /* or offline: it's time to recompute msgmni */
+  /*
+   * This is done by invoking the ipcns notifier chain with the
+   * IPC_MEMCHANGED event
+   */
+  ipcns_notify(IPCNS_MEMCHANGED);
+  break;
+ case MEM_GOING_ONLINE:
+ case MEM_GOING_OFFLINE:
+ case MEM_CANCEL_ONLINE:
+ case MEM_CANCEL_OFFLINE:
+ default:
+  break;
+ }
+
+ return NOTIFY_OK;
+}
+
+#endif /* CONFIG_MEMORY_HOTPLUG */
+
 static struct ipc_namespace *clone_ipc_ns(struct ipc_namespace *old_ns)
 {
```

```
  int err;
@@ -76,6 +104,8 @@ static struct ipc_namespace *clone_ipc_n
  if (err)
   goto err_shm;

+ register_ipcns_notifier(ns);
+
  kref_init(&ns->kref);
  return ns;

@@ -111,6 +141,15 @@ void free_ipc_ns(struct kref *kref)
  struct ipc_namespace *ns;

  ns = container_of(kref, struct ipc_namespace, kref);
+ /*
+  * Unregistering the hotplug notifier at the beginning guarantees
+  * that the ipc namespace won't be freed while we are inside the
+  * the callback routine. Since the blocking_notifier_chain_XXX
+  * routines hold a rw lock on the notifier list,
+  * unregister_ipcns_notifier() won't take the rw lock before
+  * blocking_notifier_call_chain() has released the rd lock.
+  */
+ unregister_ipcns_notifier(ns);
  sem_exit_ns(ns);
  msg_exit_ns(ns);
  shm_exit_ns(ns);
@@ -123,6 +162,9 @@ void free_ipc_ns(struct kref *kref)
  *
  * The various system5 IPC resources (semaphores, messages and shared
  * memory) are initialised
+ * A callback routine is registered into the memory hotplug notifier
+ * chain: since msgmni scales to lowmem this callback routine will be
+ * called upon successful memory add / remove to recompute msmgni.
  */

 static int __init ipc_init(void)
@@ -130,6 +172,8 @@ static int __init ipc_init(void)
  sem_init();
  msg_init();
  shm_init();
+ hotplug_memory_notifier(ipc_memory_callback, IPC_CALLBACK_PRI);
+ register_ipcns_notifier(&init_ipc_ns);
  return 0;
 }
 __initcall(ipc_init);
Index: linux-2.6.24/ipc/msg.c
===================================================================
--- linux-2.6.24.orig/ipc/msg.c 2008-01-31 10:08:49.000000000 +0100
```

```
+++ linux-2.6.24/ipc/msg.c 2008-01-31 11:06:17.000000000 +0100
@@ -86,7 +86,7 @@ static int sysvipc_msg_proc_show(struct
  * Also take into account the number of nsproxies created so far.
  * This should be done staying within the (MSGMNI , IPCMNI/nr_ipc_ns) range.
  */
-static void recompute_msgmni(struct ipc_namespace *ns)
+void recompute_msgmni(struct ipc_namespace *ns)
 {
  struct sysinfo i;
  unsigned long allowed;
Index: linux-2.6.24/ipc/util.h
===================================================================
--- linux-2.6.24.orig/ipc/util.h 2008-01-29 16:55:04.000000000 +0100
+++ linux-2.6.24/ipc/util.h 2008-01-31 11:07:10.000000000 +0100
@@ -134,6 +134,8 @@ extern int ipcget_new(struct ipc_namespa
 extern int ipcget_public(struct ipc_namespace *, struct ipc_ids *,
    struct ipc_ops *, struct ipc_params *);

+extern void recompute_msgmni(struct ipc_namespace *);
+
 static inline int ipc_buildid(int id, int seq)
 {
  return SEQ_MULTIPLIER * seq + id;

--

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

Subject: [RFC][PATCH v2 5/7] Invoke the ipcns notifier chain as a work item
Posted by Nadia Derbey on Thu, 31 Jan 2008 13:40:23 GMT
View Forum Message <> Reply to Message

[PATCH 05/07]

This patche makes the memory hotplug chain's mutex held for a shorter time:
when memory is offlined or onlined a work item is added to the global
workqueue.
When the work item is run, it notifies the ipcns notifier chain with the
IPCNS_MEMCHANGED event.

Signed-off-by: Nadia Derbey <Nadia.Derbey@bull.net>

---
 ipc/util.c | 17 ++++++++++++++--
 1 file changed, 15 insertions(+), 2 deletions(-)

```
Index: linux-2.6.24/ipc/util.c
===================================================================
--- linux-2.6.24.orig/ipc/util.c 2008-01-31 11:04:51.000000000 +0100
+++ linux-2.6.24/ipc/util.c 2008-01-31 11:41:01.000000000 +0100
@@ -57,6 +57,14 @@ atomic_t nr_ipc_ns = ATOMIC_INIT(1);

 #ifdef CONFIG_MEMORY_HOTPLUG

+static void ipc_memory_notifier(struct work_struct *work)
+{
+ ipcns_notify(IPCNS_MEMCHANGED);
+}
+
+static DECLARE_WORK(ipc_memory_wq, ipc_memory_notifier);
+
+
 static int ipc_memory_callback(struct notifier_block *self,
     unsigned long action, void *arg)
 {
@@ -65,9 +73,14 @@ static int ipc_memory_callback(struct no
  case MEM_OFFLINE:   /* or offline: it's time to recompute msgmni */
   /*
    * This is done by invoking the ipcns notifier chain with the
-   * IPC_MEMCHANGED event
+   * IPC_MEMCHANGED event.
+   * In order not to keep the lock on the hotplug memory chain
+   * for too long, queue a work item that will, when waken up,
+   * activate the ipcns notification chain.
+   * No need to keep several ipc work items on the queue.
    */
-  ipcns_notify(IPCNS_MEMCHANGED);
+  if (!work_pending(&ipc_memory_wq))
+   schedule_work(&ipc_memory_wq);
   break;
  case MEM_GOING_ONLINE:
  case MEM_GOING_OFFLINE:

--

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

Subject: [RFC][PATCH v2 6/7] Recomputing msgmni on ipc namespace creation/removal

[PATCH 06/07]

This patch introduces a notification mechanism that aims at recomputing
msgmni each time an ipc namespace is created or removed.

The ipc namespace notifier chain already defined for memory hotplug management
is used for that purpose too.

Each time a new ipc namespace is allocated or an existing ipc namespace is
removed, the ipcns notifier chain is notified. The callback routine for each
registered ipc namespace is then activated in order to recompute msgmni for
that namespace.

Signed-off-by: Nadia Derbey <Nadia.Derbey@bull.net>

---
 include/linux/ipc.h | 25 ++----------------------
 ipc/Makefile | 3 +--
 ipc/ipcns_notifier.c | 2 ++
 ipc/util.c | 12 ++++++++++++
 4 files changed, 17 insertions(+), 25 deletions(-)

Index: linux-2.6.24/include/linux/ipc.h
===
--- linux-2.6.24.orig/include/linux/ipc.h 2008-01-31 10:49:07.000000000 +0100
+++ linux-2.6.24/include/linux/ipc.h 2008-01-31 11:46:32.000000000 +0100
@@ -81,9 +81,7 @@ struct ipc_kludge {

 #include <linux/kref.h>
 #include <linux/spinlock.h>
-#ifdef CONFIG_MEMORY_HOTPLUG
 #include <linux/notifier.h>
-#endif /* CONFIG_MEMORY_HOTPLUG */

 #define IPCMNI 32768 /* <= MAX_INT limit for ipc arrays (including sysctl changes) */

@@ -92,6 +90,8 @@ struct ipc_kludge {
  * ipc namespace events
  */
 #define IPCNS_MEMCHANGED   0x00000001   /* Notify lowmem size changed */
+#define IPCNS_CREATED 0x00000002   /* Notify new ipc namespace created */
+#define IPCNS_REMOVED 0x00000003   /* Notify ipc namespace removed */

 #define IPCNS_CALLBACK_PRI 0

```
@@ -130,9 +130,7 @@ struct ipc_namespace {
  int  shm_ctlmni;
  int  shm_tot;

-#ifdef CONFIG_MEMORY_HOTPLUG
  struct notifier_block ipcns_nb;
-#endif
 };

 extern struct ipc_namespace init_ipc_ns;
@@ -143,29 +141,10 @@ extern atomic_t nr_ipc_ns;
 extern void free_ipc_ns(struct kref *kref);
 extern struct ipc_namespace *copy_ipcs(unsigned long flags,
       struct ipc_namespace *ns);
-#ifdef CONFIG_MEMORY_HOTPLUG
-
 extern int register_ipcns_notifier(struct ipc_namespace *);
 extern int unregister_ipcns_notifier(struct ipc_namespace *);
 extern int ipcns_notify(unsigned long);

-#else /* CONFIG_MEMORY_HOTPLUG */
-
-static inline int register_ipcns_notifier(struct ipc_namespace *ipcns)
-{
- return 0;
-}
-static inline int unregister_ipcns_notifier(struct ipc_namespace *ipcns)
-{
- return 0;
-}
-static inline int ipcns_notify(unsigned long ev)
-{
- return 0;
-}
-
-#endif /* CONFIG_MEMORY_HOTPLUG */
-
 #else
 #define INIT_IPC_NS(ns)
 static inline struct ipc_namespace *copy_ipcs(unsigned long flags,
Index: linux-2.6.24/ipc/ipcns_notifier.c
===================================================================
--- linux-2.6.24.orig/ipc/ipcns_notifier.c 2008-01-31 10:58:11.000000000 +0100
+++ linux-2.6.24/ipc/ipcns_notifier.c 2008-01-31 11:48:32.000000000 +0100
@@ -27,6 +27,8 @@ static int ipcns_callback(struct notifie

  switch (action) {
  case IPCNS_MEMCHANGED:   /* amount of lowmem has changed */
```

```
+ case IPCNS_CREATED:
+ case IPCNS_REMOVED:
   /*
    * It's time to recompute msgmni
    */
Index: linux-2.6.24/ipc/Makefile
===================================================================
--- linux-2.6.24.orig/ipc/Makefile 2008-01-31 10:59:39.000000000 +0100
+++ linux-2.6.24/ipc/Makefile 2008-01-31 11:49:13.000000000 +0100
@@ -3,8 +3,7 @@
 #

 obj-$(CONFIG_SYSVIPC_COMPAT) += compat.o
-obj_mem-$(CONFIG_MEMORY_HOTPLUG) += ipcns_notifier.o
-obj-$(CONFIG_SYSVIPC) += util.o msgutil.o msg.o sem.o shm.o $(obj_mem-y)
+obj-$(CONFIG_SYSVIPC) += util.o msgutil.o msg.o sem.o shm.o ipcns_notifier.o
 obj-$(CONFIG_SYSVIPC_SYSCTL) += ipc_sysctl.o
 obj_mq-$(CONFIG_COMPAT) += compat_mq.o
 obj-$(CONFIG_POSIX_MQUEUE) += mqueue.o msgutil.o $(obj_mq-y)
Index: linux-2.6.24/ipc/util.c
===================================================================
--- linux-2.6.24.orig/ipc/util.c 2008-01-31 11:41:01.000000000 +0100
+++ linux-2.6.24/ipc/util.c 2008-01-31 12:00:46.000000000 +0100
@@ -117,6 +117,12 @@ static struct ipc_namespace *clone_ipc_n
  if (err)
   goto err_shm;

+ /*
+  * msgmni has already been computed for the new ipc ns.
+  * Thus, do the ipcns creation notification before registering that
+  * new ipcns in the chain.
+  */
+ ipcns_notify(IPCNS_CREATED);
  register_ipcns_notifier(ns);

  kref_init(&ns->kref);
@@ -168,6 +174,12 @@ void free_ipc_ns(struct kref *kref)
  shm_exit_ns(ns);
  kfree(ns);
  atomic_dec(&nr_ipc_ns);
+
+ /*
+  * Do the ipcns removal notification after decrementing nr_ipc_ns in
+  * order to have a correct value when recomputing msgmni.
+  */
+ ipcns_notify(IPCNS_REMOVED);
 }
```

/**

--

_____

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

---

Subject: [RFC][PATCH v2 7/7] Do not recompute msgmni anymore if explicitely set by user
Posted by Nadia Derbey on Thu, 31 Jan 2008 13:40:25 GMT
View Forum Message <> Reply to Message

[PATCH 07/07]

This patch makes msgmni not recomputed anymore upon ipc namespace creation / removal or memory add/remove, as soon as it has been set from userland.

As soon as msgmni is explicitely set via procfs or sysctl(), the associated callback routine is unregistered from the ipc namespace notifier chain.

Signed-off-by: Nadia Derbey <Nadia.Derbey@bull.net>

---
ipc/ipc_sysctl.c | 43 ++---
1 file changed, 41 insertions(+), 2 deletions(-)

Index: linux-2.6.24/ipc/ipc_sysctl.c
===
--- linux-2.6.24.orig/ipc/ipc_sysctl.c 2008-01-29 16:55:04.000000000 +0100
+++ linux-2.6.24/ipc/ipc_sysctl.c 2008-01-31 13:13:14.000000000 +0100
@@ -34,6 +34,24 @@ static int proc_ipc_dointvec(ctl_table *
 return proc_dointvec(&ipc_table, write, filp, buffer, lenp, ppos);
 }

+static int proc_ipc_callback_dointvec(ctl_table *table, int write,
+ struct file *filp, void __user *buffer, size_t *lenp, loff_t *ppos)
+{
+ size_t lenp_bef = *lenp;
+ int rc;
+
+ rc = proc_ipc_dointvec(table, write, filp, buffer, lenp, ppos);
+
+ if (write && !rc && lenp_bef == *lenp)
+ /*
+ * Tunable has successfully been changed from userland:

```
+   * disable its automatic recomputing.
+   */
+  unregister_ipcns_notifier(current->nsproxy->ipc_ns);
+
+ return rc;
+}
+
 static int proc_ipc_doulongvec_minmax(ctl_table *table, int write,
   struct file *filp, void __user *buffer, size_t *lenp, loff_t *ppos)
 {
@@ -48,6 +66,7 @@ static int proc_ipc_doulongvec_minmax(ct
 #else
 #define proc_ipc_doulongvec_minmax NULL
 #define proc_ipc_dointvec     NULL
+#define proc_ipc_callback_dointvec NULL
 #endif

 #ifdef CONFIG_SYSCTL_SYSCALL
@@ -89,8 +108,28 @@ static int sysctl_ipc_data(ctl_table *ta
   }
   return 1;
 }
+
+static int sysctl_ipc_registered_data(ctl_table *table, int __user *name,
+  int nlen, void __user *oldval, size_t __user *oldlenp,
+  void __user *newval, size_t newlen)
+{
+ int rc;
+
+ rc = sysctl_ipc_data(table, name, nlen, oldval, oldlenp, newval,
+  newlen);
+
+ if (newval && newlen && rc > 0)
+  /*
+   * Tunable has successfully been changed from userland:
+   * disable its automatic recomputing.
+   */
+  unregister_ipcns_notifier(current->nsproxy->ipc_ns);
+
+ return rc;
+}
 #else
 #define sysctl_ipc_data NULL
+#define sysctl_ipc_registered_data NULL
 #endif

 static struct ctl_table ipc_kern_table[] = {
@@ -136,8 +175,8 @@ static struct ctl_table ipc_kern_table[]
```

```
   .data  = &init_ipc_ns.msg_ctlmni,
   .maxlen  = sizeof (init_ipc_ns.msg_ctlmni),
   .mode  = 0644,
-  .proc_handler = proc_ipc_dointvec,
-  .strategy = sysctl_ipc_data,
+  .proc_handler = proc_ipc_callback_dointvec,
+  .strategy = sysctl_ipc_registered_data,
  },
  {
   .ctl_name = KERN_MSGMNB,

--
```

Subject: Re: [RFC][PATCH v2 7/7] Do not recompute msgmni anymore if explicitely set by user
Posted by Yasunori Goto on Tue, 05 Feb 2008 13:38:39 GMT
View Forum Message <> Reply to Message

Thanks Nadia-san.

I tested this patch set on my box. It works well.
I have only one comment.

```
> ---
>  ipc/ipc_sysctl.c |   43 ++++++++++++++++++++++++++++++++++++++++++--
>  1 file changed, 41 insertions(+), 2 deletions(-)
>
> Index: linux-2.6.24/ipc/ipc_sysctl.c
> ===================================================================
> --- linux-2.6.24.orig/ipc/ipc_sysctl.c 2008-01-29 16:55:04.000000000 +0100
> +++ linux-2.6.24/ipc/ipc_sysctl.c 2008-01-31 13:13:14.000000000 +0100
> @@ -34,6 +34,24 @@ static int proc_ipc_dointvec(ctl_table *
>   return proc_dointvec(&ipc_table, write, filp, buffer, lenp, ppos);
>  }
>
> +static int proc_ipc_callback_dointvec(ctl_table *table, int write,
> + struct file *filp, void __user *buffer, size_t *lenp, loff_t *ppos)
> +{
> + size_t lenp_bef = *lenp;
> + int rc;
> +
> + rc = proc_ipc_dointvec(table, write, filp, buffer, lenp, ppos);
```

```
> +
> + if (write && !rc && lenp_bef == *lenp)
> +  /*
> +   * Tunable has successfully been changed from userland:
> +   * disable its automatic recomputing.
> +   */
> +  unregister_ipcns_notifier(current->nsproxy->ipc_ns);
> +
> + return rc;
> +}
> +
```

Hmmm. I suppose this may be side effect which user does not wish.

I would like to recommend there should be a switch which can turn on/off
automatic recomputing.
If user would like to change this value, it should be turned off.
Otherwise, his requrest will be rejected with some messages.

Probably, user can understand easier than this side effect.

Bye.

--
Yasunori Goto

Subject: Re: [RFC][PATCH v2 7/7] Do not recompute msgmni anymore if explicitely set by user
Posted by Nadia Derbey on Tue, 05 Feb 2008 14:55:10 GMT
View Forum Message <> Reply to Message

Yasunori Goto wrote:
> Thanks Nadia-san.
>
> I tested this patch set on my box. It works well.
> I have only one comment.
>
>
>
>>>---

>> ipc/ipc_sysctl.c | 43 ++--
>> 1 file changed, 41 insertions(+), 2 deletions(-)
>>
>>Index: linux-2.6.24/ipc/ipc_sysctl.c
>>===
>>--- linux-2.6.24.orig/ipc/ipc_sysctl.c 2008-01-29 16:55:04.000000000 +0100
>>+++ linux-2.6.24/ipc/ipc_sysctl.c 2008-01-31 13:13:14.000000000 +0100
>>@@ -34,6 +34,24 @@ static int proc_ipc_dointvec(ctl_table *
>> return proc_dointvec(&ipc_table, write, filp, buffer, lenp, ppos);
>> }
>>
>>+static int proc_ipc_callback_dointvec(ctl_table *table, int write,
>>+ struct file *filp, void __user *buffer, size_t *lenp, loff_t *ppos)
>>+{
>>+ size_t lenp_bef = *lenp;
>>+ int rc;
>>+
>>+ rc = proc_ipc_dointvec(table, write, filp, buffer, lenp, ppos);
>>+
>>+ if (write && !rc && lenp_bef == *lenp)
>>+ /*
>>+   * Tunable has successfully been changed from userland:
>>+   * disable its automatic recomputing.
>>+   */
>>+ unregister_ipcns_notifier(current->nsproxy->ipc_ns);
>>+
>>+ return rc;
>>+}
>>+
>
>
>
> Hmmm. I suppose this may be side effect which user does not wish.
>
> I would like to recommend there should be a switch which can turn on/off
> automatic recomputing.
> If user would like to change this value, it should be turned off.
> Otherwise, his requrest will be rejected with some messages.
>
> Probably, user can understand easier than this side effect.
>

Hi Yasunori,

Hope you're feeling better!

Well the idea behind this was the following: if msgmni is changed say
via procfs it is usually to increase it, in order for applications that

need more msg queues to be able to run.
So even if a new ipc namespace is created or memory is removed, the application that has set that new value doesn't care: it wants msgmni to be unchanged. I agree with you that unconditionally turning the recomputing off may appear coarse, but I'm afraid that adding the functionality of turning that recomputing on/off will make things more complicated:
1) manage the tunable recomputing state: it shouldn't be turned on twice
2) adds more things to do at the user level.

I'll try to think about it more deeply and may be come up with an intermediate solution.

Regards,
Nadia

_____