Subject: Re: [PATCH 2.6.24-rc8-mm1 12/15] (RFC) IPC/semaphores: make use of RCU to free the sem_undo_list
Posted by serue on Wed, 30 Jan 2008 21:26:50 GMT
View Forum Message <> Reply to Message

Quoting pierre.peiffer@bull.net (pierre.peiffer@bull.net):
> From: Pierre Peiffer <pierre.peiffer@bull.net>
>
> Today, the sem_undo_list is freed when the last task using it exits.
> There is no mechanism in place, that allows a safe concurrent access to
> the sem_undo_list of a target task and protects efficiently against a
> task-exit.
>
> That is okay for now as we don't need this.
>
> As I would like to provide a /proc interface to access this data, I need
> such a safe access, without blocking the target task if possible.
>
> This patch proposes to introduce the use of RCU to delay the real free of
> these sem_undo_list structures. They can then be accessed in a safe manner
> by any tasks inside read critical section, this way:
>
> struct sem_undo_list *undo_list;
> int ret;
> ...
> rcu_read_lock();
> undo_list = rcu_dereference(task->sysvsem.undo_list);
> if (undo_list)
>   ret = atomic_inc_not_zero(&undo_list->refcnt);
> rcu_read_unlock();
> ...
> if (undo_list && ret) {
>   /* section where undo_list can be used quietly */
>   ...
> }
> ...

And of course then

 if (atomic_dec_and_test(&undo_list->refcnt))
  free_semundo_list(undo_list);

by that task.

>
> Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>

Looks correct in terms of locking/refcounting.

Signed-off-by: Serge Hallyn <serue@us.ibm.com>

thanks,
-serge

> ---
>
> include/linux/sem.h | 7 +++++--
> ipc/sem.c | 42 ++++++++++++++++++++++++++++++++----------
> 2 files changed, 31 insertions(+), 18 deletions(-)
>
> Index: b/include/linux/sem.h
> ===
> --- a/include/linux/sem.h
> +++ b/include/linux/sem.h
> @@ -115,7 +115,8 @@ struct sem_queue {
> };
>
> /* Each task has a list of undo requests. They are executed automatically
> - * when the process exits.
> + * when the last refcnt of sem_undo_list is released (ie when the process exits
> + * in the general case)
>   */
> struct sem_undo {
>   struct sem_undo * proc_next; /* next entry on this process */
> @@ -125,12 +126,14 @@ struct sem_undo {
> };
>
> /* sem_undo_list controls shared access to the list of sem_undo structures
> - * that may be shared among all a CLONE_SYSVSEM task group.
> + * that may be shared among all a CLONE_SYSVSEM task group or with an external
> + * process which changes the list through procfs.
>   */
> struct sem_undo_list {
>   atomic_t refcnt;
>   spinlock_t lock;
>   struct sem_undo *proc_list;
> + struct ipc_namespace *ns;
> };
>
> struct sysv_sem {
> Index: b/ipc/sem.c
> ===
> --- a/ipc/sem.c
> +++ b/ipc/sem.c
> @@ -1038,6 +1038,7 @@ static inline int get_undo_list(struct s
>   return -ENOMEM;

```
>    spin_lock_init(&undo_list->lock);
>    atomic_set(&undo_list->refcnt, 1);
> +  undo_list->ns = get_ipc_ns(current->nsproxy->ipc_ns);
>    current->sysvsem.undo_list = undo_list;
>   }
>   *undo_listp = undo_list;
> @@ -1316,7 +1317,8 @@ int copy_semundo(unsigned long clone_fla
>  }
>
>  /*
> - * add semadj values to semaphores, free undo structures.
> + * add semadj values to semaphores, free undo structures, if there is no
> + * more user.
>   * undo structures are not freed when semaphore arrays are destroyed
>   * so some of them may be out of date.
>   * IMPLEMENTATION NOTE: There is some confusion over whether the
> @@ -1326,23 +1328,17 @@ int copy_semundo(unsigned long clone_fla
>   * The original implementation attempted to do this (queue and wait).
>   * The current implementation does not do so. The POSIX standard
>   * and SVID should be consulted to determine what behavior is mandated.
> + *
> + * Note:
> + * A concurrent task is only allowed to access and go through the list
> + * of sem_undo if it successfully grabs a refcnt.
>   */
> -void exit_sem(struct task_struct *tsk)
> +static void free_semundo_list(struct sem_undo_list *undo_list)
>  {
> - struct sem_undo_list *undo_list;
>   struct sem_undo *u, **up;
> - struct ipc_namespace *ns;
>
> - undo_list = tsk->sysvsem.undo_list;
> - if (!undo_list)
> -  return;
> -
> - if (!atomic_dec_and_test(&undo_list->refcnt))
> -  return;
> -
> - ns = tsk->nsproxy->ipc_ns;
> - /* There's no need to hold the semundo list lock, as current
> -         * is the last task exiting for this undo list.
> + /* There's no need to hold the semundo list lock, as there are
> +  * no more tasks or possible users for this undo list.
>    */
>   for (up = &undo_list->proc_list; (u = *up); *up = u->proc_next, kfree(u)) {
>    struct sem_array *sma;
> @@ -1354,7 +1350,7 @@ void exit_sem(struct task_struct *tsk)
```

```
>
>    if(semid == -1)
>     continue;
> -  sma = sem_lock(ns, semid);
> +  sma = sem_lock(undo_list->ns, semid);
>    if (IS_ERR(sma))
>     continue;
>
> @@ -1368,7 +1364,8 @@ void exit_sem(struct task_struct *tsk)
>     if (u == un)
>      goto found;
>    }
> -  printk ("exit_sem undo list error id=%d\n", u->semid);
> +  printk(KERN_ERR "free_semundo_list error id=%d\n",
> +         u->semid);
>    goto next_entry;
>  found:
>    *unp = un->id_next;
> @@ -1404,9 +1401,22 @@ found:
>  next_entry:
>    sem_unlock(sma);
>   }
> + put_ipc_ns(undo_list->ns);
>   kfree(undo_list);
>  }
>
> +/* called from do_exit() */
> +void exit_sem(struct task_struct *tsk)
> +{
> + struct sem_undo_list *ul = tsk->sysvsem.undo_list;
> + if (ul) {
> +  rcu_assign_pointer(tsk->sysvsem.undo_list, NULL);
> +  synchronize_rcu();
> +  if (atomic_dec_and_test(&ul->refcnt))
> +   free_semundo_list(ul);
> + }
> +}
> +
>  #ifdef CONFIG_PROC_FS
>  static int sysvipc_sem_proc_show(struct seq_file *s, void *it)
>  {
>
> --
> Pierre Peiffer
> _____________________________________________________
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containers
```

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