Subject: [RFC PATCH 0/5] Resend -v2 - Use procfs to change a syscall behavior
Posted by Nadia Derbey on Tue, 08 Jul 2008 11:24:22 GMT
View Forum Message <> Reply to Message

Resending after fixing the issues pointed out by Serge.

Also ported to 2.6.26-rc8-mm1.

Regards,
Nadia

This patchset is a part of an effort to change some syscalls behavior for checkpoint restart.

When restarting an object that has previously been checkpointed, its state should be unchanged compared to the checkpointed image.
For example, a restarted process should have the same upid nr as the one it used to have when being checkpointed; an ipc object should have the same id as the one it had when the checkpoint occured.
Also, talking about system V ipcs, they should be restored with the same state (e.g. in terms of pid of last operation).

This means that several syscalls should not behave in a default mode when they are called during a restart phase.

One solution consists in defining a new syscall for each syscall that is called during restart:
. sys_fork_with_id() would fork a process with a predefined id.
. sys_msgget_with_id() would create a msg queue with a predefined id
. sys_semget_with_id() would create a semaphore set with a predefined id
. etc,

This solution requires defining a new syscall each time we need an existing syscall to behave in a non-default way.

An alternative to this solution consists in defining a new field in the task structure (let's call it next_syscall_data) that, if set, would change the behavior of next syscall to be called. The sys_fork_with_id() previously cited can be replaced by
1) set next_syscall_data to a target upid nr
2) call fork().

This patch series implements the 2nd solution. Actually I've already sent it some times ago, and things ended up with Pavel complaining about the "ugly interface" (see https://lists.linux-foundation.org/pipermail/containers/2008-April/010909.html).

Now, I'm resending the series because this 2nd solution has the advantage of being easily reusable for many subsystems: the only thing needed is just to set a field in the task structure and rewrite the code portion that is sensitive to this field being set (it's successfully being used in cryo code - git tree at git://git.sr71.net/~hallyn/cryodev.git).

The patches have been ported to 2.6.26-rc8-mm1 and the open() syscall in now covered.

A new file is created in procfs: /proc/self/task/<my_tid>/next_syscall_data. This makes it possible to avoid races between several threads belonging to the same process.

Setting a value into this file fills in the next_syscall_data in the task structure.

The following subsystems have been changed to take this value into account:
1) sysvipc:
  . if there's a value in next_syscall_data when msgget() is called, msgget() creates a msg queue with that value as an id
  . this applies to semget() and shmget().
  . if next_syscall_data is set to 1 when msgctl(IPC_SET) is called, msgctl() sets more that the usual permission fields for the target msg queue (it sets the time fields, and the pid of last operation fields).
  . this applies to semctl() and shmctl().
2) process creation:
  . if there's a value in next_syscall_data when fork() is called, fork() creates a process with that value as a pid.
  . this applies to vfork() and clone().
3) file descriptors:
  . if there's a value in next_syscall_data when open() is called, open() uses that value as the file descriptor for the open file

The syntax is:
echo "LONG1 XX" > /proc/self/task/<my_tid>/next_syscall_data
    next object to be created will have an id set to XX

Today, the ids are specified as long, but having a type string specified in the next_syscall_data file makes it possible to cover more types in the future, if needed.
Also, only a single value can be set. But the number that immediatly follows the type string makes it possible to specify more values in the future, if needed. This can be applied, e.g. to predefine all the upid nrs for a process that belongs to nested namespaces, if needed in the future.

These patches should be applied to 2.6.26-rc8-mm1, in the following order:

[PATCH 1/5] : next_syscall_data_proc_file.patch
[PATCH 2/5] : ipccreate_use_next_syscall_data.patch
[PATCH 3/5] : proccreate_use_next_syscall_data.patch
[PATCH 4/5] : ipcset_use_next_syscall_data.patch
[PATCH 5/5] : fileopen_use_next_syscall_data.patch

Any comment and/or suggestions are welcome.

Regards,
Nadia

--

Subject: [RFC PATCH 1/5] adds the procfs facilities
Posted by Nadia Derbey on Tue, 08 Jul 2008 11:24:23 GMT
View Forum Message <> Reply to Message

[PATCH 01/05]

This patch adds the procfs facility needed to feed some data for the
next syscall to be called.

The effect of issuing
echo "LONG<Y> <XX>" > /proc/self/task/<tid>/next_syscall_data
is that <XX> will be stored in a new field of the task structure
(next_syscall_data). This field, in turn will be taken as the data to feed
next syscall that supports the feature.

<Y> is the number of values provided on the line.
For the sake of simplicity it is now fixed to 1, but this can be extended as
needed, in the future.

This is particularly useful when restarting an application, as we need
sometimes the syscalls to have a non-default behavior.

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

```
 fs/exec.c                    |    6 +
 fs/proc/base.c               |   75 ++++++++++++++++++++
 include/linux/next_syscall_data.h |   32 ++++++++
 include/linux/sched.h        |    6 +
```

```
 kernel/Makefile                 |    3
 kernel/exit.c                   |    4 +
 kernel/fork.c                   |    2
 kernel/next_syscall_data.c      |  151 ++++++++++++++++++++++++++++++++++++++++
 8 files changed, 278 insertions(+), 1 deletion(-)

Index: linux-2.6.26-rc8-mm1/include/linux/sched.h
===================================================================
--- linux-2.6.26-rc8-mm1.orig/include/linux/sched.h 2008-07-08 09:04:21.000000000 +0200
+++ linux-2.6.26-rc8-mm1/include/linux/sched.h 2008-07-08 09:13:43.000000000 +0200
@@ -87,6 +87,7 @@ struct sched_param {
 #include <linux/task_io_accounting.h>
 #include <linux/kobject.h>
 #include <linux/latencytop.h>
+#include <linux/next_syscall_data.h>

 #include <asm/processor.h>

@@ -1296,6 +1297,11 @@ struct task_struct {
  int latency_record_count;
  struct latency_record latency_record[LT_SAVECOUNT];
 #endif
+ /*
+  * If non-NULL indicates that next operation will be forced, e.g.
+  * that next object to be created will have a predefined id.
+  */
+ struct next_syscall_data *nsd;
 };

 /*
Index: linux-2.6.26-rc8-mm1/include/linux/next_syscall_data.h
===================================================================
--- /dev/null 1970-01-01 00:00:00.000000000 +0000
+++ linux-2.6.26-rc8-mm1/include/linux/next_syscall_data.h 2008-07-08 09:24:38.000000000
+0200
@@ -0,0 +1,32 @@
+/*
+ * include/linux/next_syscall_data.h
+ *
+ * Definitions to support fixed data for next syscall to be called.
+ */
+
+#ifndef _LINUX_NEXT_SYSCALL_DATA_H
+#define _LINUX_NEXT_SYSCALL_DATA_H
+
+#define NDATA 1
+
+/*
```

```
+ * If this structure is pointed to by a task_struct, next syscall to be called
+ * by the task will have a non-default behavior.
+ * For example, it can be used to pre-set the id of the object to be created
+ * by next syscall.
+ */
+struct next_syscall_data {
+ int ndata;
+ long data[NDATA];
+};
+
+extern ssize_t get_next_syscall_data(struct task_struct *, char *, size_t);
+extern int set_next_syscall_data(struct task_struct *, char *);
+extern void reset_next_syscall_data(struct task_struct *);
+
+static inline void exit_next_syscall_data(struct task_struct *tsk)
+{
+ reset_next_syscall_data(tsk);
+}
+
+#endif /* _LINUX_NEXT_SYSCALL_DATA_H */
Index: linux-2.6.26-rc8-mm1/fs/proc/base.c
===================================================================
--- linux-2.6.26-rc8-mm1.orig/fs/proc/base.c 2008-07-08 09:05:13.000000000 +0200
+++ linux-2.6.26-rc8-mm1/fs/proc/base.c 2008-07-08 09:18:12.000000000 +0200
@@ -1158,6 +1158,76 @@ static const struct file_operations proc
 };
 #endif

+static ssize_t next_syscall_data_read(struct file *file, char __user *buf,
+    size_t count, loff_t *ppos)
+{
+ struct task_struct *task;
+ char *page;
+ ssize_t length;
+
+ task = get_proc_task(file->f_path.dentry->d_inode);
+ if (!task)
+  return -ESRCH;
+
+ if (count >= PAGE_SIZE)
+  count = PAGE_SIZE - 1;
+
+ length = -ENOMEM;
+ page = (char *) __get_free_page(GFP_TEMPORARY);
+ if (!page)
+  goto out;
+
+ length = get_next_syscall_data(task, (char *) page, count);
```

```
+ if (length >= 0)
+  length = simple_read_from_buffer(buf, count, ppos,
+      (char *)page, length);
+ free_page((unsigned long) page);
+
+out:
+ put_task_struct(task);
+ return length;
+}
+
+static ssize_t next_syscall_data_write(struct file *file,
+    const char __user *buf,
+    size_t count, loff_t *ppos)
+{
+ struct inode *inode = file->f_path.dentry->d_inode;
+ char *page;
+ ssize_t length;
+
+ if (pid_task(proc_pid(inode), PIDTYPE_PID) != current)
+  return -EPERM;
+
+ if (count >= PAGE_SIZE)
+  count = PAGE_SIZE - 1;
+
+ if (*ppos != 0) {
+  /* No partial writes. */
+  return -EINVAL;
+ }
+ page = (char *)__get_free_page(GFP_TEMPORARY);
+ if (!page)
+  return -ENOMEM;
+ length = -EFAULT;
+ if (copy_from_user(page, buf, count))
+  goto out_free_page;
+
+ page[count] = '\0';
+
+ length = set_next_syscall_data(current, page);
+ if (!length)
+  length = count;
+
+out_free_page:
+ free_page((unsigned long) page);
+ return length;
+}
+
+static const struct file_operations proc_next_syscall_data_operations = {
+ .read  = next_syscall_data_read,
```

```
+ .write  = next_syscall_data_write,
+};

 #ifdef CONFIG_SCHED_DEBUG
 /*
@@ -2853,6 +2923,11 @@ static const struct pid_entry tid_base_s
 #ifdef CONFIG_TASK_IO_ACCOUNTING
  INF("io", S_IRUGO, tid_io_accounting),
 #endif
+ /*
+  * NOTE that this file is not added into tgid_base_stuff[] since it
+  * has to be specified on a per-thread basis.
+  */
+ REG("next_syscall_data", S_IRUGO|S_IWUSR, next_syscall_data),
 };

 static int proc_tid_base_readdir(struct file * filp,
Index: linux-2.6.26-rc8-mm1/kernel/Makefile
===================================================================
--- linux-2.6.26-rc8-mm1.orig/kernel/Makefile 2008-07-08 09:04:35.000000000 +0200
+++ linux-2.6.26-rc8-mm1/kernel/Makefile 2008-07-08 09:19:14.000000000 +0200
@@ -9,7 +9,8 @@ obj-y     = sched.o fork.o exec_domain.o
       rcupdate.o extable.o params.o posix-timers.o \
       kthread.o wait.o kfifo.o sys_ni.o posix-cpu-timers.o mutex.o \
       hrtimer.o rwsem.o nsproxy.o srcu.o semaphore.o \
-      notifier.o ksysfs.o pm_qos_params.o sched_clock.o
+      notifier.o ksysfs.o pm_qos_params.o sched_clock.o \
+      next_syscall_data.o

 CFLAGS_REMOVE_sched.o = -pg -mno-spe

Index: linux-2.6.26-rc8-mm1/kernel/next_syscall_data.c
===================================================================
--- /dev/null 1970-01-01 00:00:00.000000000 +0000
+++ linux-2.6.26-rc8-mm1/kernel/next_syscall_data.c 2008-07-08 09:35:27.000000000 +0200
@@ -0,0 +1,151 @@
+/*
+ * linux/kernel/next_syscall_data.c
+ *
+ *
+ * Provide the get_next_syscall_data() / set_next_syscall_data() routines
+ * (called from fs/proc/base.c).
+ * They allow to specify some particular data for the next syscall to be
+ * called.
+ * E.g. they can be used to specify the id for the next resource to be
+ * allocated, instead of letting the allocator set it for us.
+ */
+
```

```
+#include <linux/sched.h>
+#include <linux/ctype.h>
+
+
+
+ssize_t get_next_syscall_data(struct task_struct *task, char *buffer,
+    size_t size)
+{
+ struct next_syscall_data *nsd;
+ char *bufptr = buffer;
+ ssize_t rc, count = 0;
+ int i;
+
+ nsd = task->nsd;
+ if (!nsd || !nsd->ndata)
+  return snprintf(buffer, size, "UNSET\n");
+
+ count = snprintf(bufptr, size, "LONG%d ", nsd->ndata);
+
+ for (i = 0; i < nsd->ndata - 1; i++) {
+  rc = snprintf(&bufptr[count], size - count, "%ld ",
+    nsd->data[i]);
+  if (rc >= size - count)
+   return -ENOMEM;
+  count += rc;
+ }
+
+ rc = snprintf(&bufptr[count], size - count, "%ld\n", nsd->data[i]);
+ if (rc >= size - count)
+  return -ENOMEM;
+ count += rc;
+
+ return count;
+}
+
+static int fill_next_syscall_data(struct task_struct *task, int ndata,
+    char *buffer)
+{
+ char *token, *buff = buffer;
+ char *end;
+ struct next_syscall_data *nsd = task->nsd;
+ int i;
+
+ if (!nsd) {
+  nsd = kmalloc(sizeof(*nsd), GFP_KERNEL);
+  if (!nsd)
+   return -ENOMEM;
+  task->nsd = nsd;
```

```
+ }
+
+ nsd->ndata = ndata;
+
+ i = 0;
+ while ((token = strsep(&buff, " ")) != NULL && i < ndata) {
+  long data;
+
+  if (!*token)
+   goto out_free;
+  data = simple_strtol(token, &end, 0);
+  if (end == token || (*end && !isspace(*end)))
+   goto out_free;
+  nsd->data[i] = data;
+  i++;
+ }
+
+ if (i != ndata)
+  goto out_free;
+
+ return 0;
+
+out_free:
+ kfree(nsd);
+ task->nsd = NULL;
+ return -EINVAL;
+}
+
+/*
+ * Parses a line with the following format:
+ * <x> <id0> ... <idx-1>
+ * Currently, only x=1 is accepted.
+ * Any trailing character on the line is skipped.
+ */
+static int do_set_next_syscall_data(struct task_struct *task, char *nb,
+     char *buffer)
+{
+ int ndata;
+ char *end;
+
+ ndata = simple_strtol(nb, &end, 0);
+ if (*end)
+  return -EINVAL;
+
+ if (ndata > NDATA)
+  return -EINVAL;
+
+ return fill_next_syscall_data(task, ndata, buffer);
```

```
+}
+
+void reset_next_syscall_data(struct task_struct *task)
+{
+ struct next_syscall_data *nsd = task->nsd;
+
+ if (nsd) {
+  task->nsd = NULL;
+  kfree(nsd);
+ }
+}
+
+#define LONG_STR "LONG"
+#define RESET_STR "RESET"
+
+/*
+ * Parses a line written to /proc/self/task/<my_tid>/next_syscall_data.
+ * this line has the following format:
+ * LONG<x> id              --> a sequence of id(s) is specified
+ *                         currently, only x=1 is accepted
+ */
+int set_next_syscall_data(struct task_struct *task, char *buffer)
+{
+ char *token, *out = buffer;
+ size_t sz;
+
+ if (!out)
+  return -EINVAL;
+
+ token = strsep(&out, " ");
+
+ sz = strlen(LONG_STR);
+
+ if (!strncmp(token, LONG_STR, sz))
+  return do_set_next_syscall_data(task, token + sz, out);
+
+ if (!strncmp(token, RESET_STR, strlen(RESET_STR))) {
+  reset_next_syscall_data(task);
+  return 0;
+ }
+
+ return -EINVAL;
+}
Index: linux-2.6.26-rc8-mm1/kernel/fork.c
===================================================================
--- linux-2.6.26-rc8-mm1.orig/kernel/fork.c 2008-07-08 09:04:35.000000000 +0200
+++ linux-2.6.26-rc8-mm1/kernel/fork.c 2008-07-08 09:25:35.000000000 +0200
@@ -1085,6 +1085,8 @@ static struct task_struct *copy_process(
```

```
  p->blocked_on = NULL; /* not blocked yet */
 #endif

+ p->nsd = NULL; /* no next syscall data is the default */
+
  /* Perform scheduler related setup. Assign this task to a CPU. */
  sched_fork(p, clone_flags);

Index: linux-2.6.26-rc8-mm1/fs/exec.c
===================================================================
--- linux-2.6.26-rc8-mm1.orig/fs/exec.c 2008-07-08 09:05:13.000000000 +0200
+++ linux-2.6.26-rc8-mm1/fs/exec.c 2008-07-08 09:26:21.000000000 +0200
@@ -1016,6 +1016,12 @@ int flush_old_exec(struct linux_binprm *
  flush_signal_handlers(current, 0);
  flush_old_files(current->files);

+ /*
+  * the next syscall data is not inherited across execve()
+  */
+ if (unlikely(current->nsd))
+  reset_next_syscall_data(current);
+
  return 0;

 out:
Index: linux-2.6.26-rc8-mm1/kernel/exit.c
===================================================================
--- linux-2.6.26-rc8-mm1.orig/kernel/exit.c 2008-07-08 09:04:35.000000000 +0200
+++ linux-2.6.26-rc8-mm1/kernel/exit.c 2008-07-08 09:27:31.000000000 +0200
@@ -1066,6 +1066,10 @@ NORET_TYPE void do_exit(long code)

  proc_exit_connector(tsk);
  exit_notify(tsk, group_dead);
+
+ if (unlikely(tsk->nsd))
+  exit_next_syscall_data(tsk);
+
 #ifdef CONFIG_NUMA
  mpol_put(tsk->mempolicy);
  tsk->mempolicy = NULL;

--
```

Subject: [RFC PATCH 2/5] use next syscall data to predefine ipc objects ids
Posted by Nadia Derbey on Tue, 08 Jul 2008 11:24:24 GMT
View Forum Message <> Reply to Message

[PATCH 02/05]

This patch uses the value written into the next_syscall_data proc file
as a target id for the next IPC object to be created.
The following syscalls have a new behavior if next_syscall_data is set:
. mssget()
. semget()
. shmget()

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

---
 include/linux/next_syscall_data.h | 17 +++++++++++++--
 ipc/util.c | 39 ++++++++++++++++++++++++++++++++++++++--------
 2 files changed, 46 insertions(+), 10 deletions(-)

Index: linux-2.6.26-rc8-mm1/include/linux/next_syscall_data.h
===
--- linux-2.6.26-rc8-mm1.orig/include/linux/next_syscall_data.h 2008-07-08 09:24:38.000000000 +0200
+++ linux-2.6.26-rc8-mm1/include/linux/next_syscall_data.h 2008-07-08 12:12:39.000000000 +0200
@@ -1,7 +1,10 @@
 /*
  * include/linux/next_syscall_data.h
  *
- * Definitions to support fixed data for next syscall to be called.
+ * Definitions to support fixed data for next syscall to be called. The
+ * following is supported today:
+ *   . object creation with a predefined id
+ *   . for a sysv ipc object
  */

 #ifndef _LINUX_NEXT_SYSCALL_DATA_H
@@ -13,13 +16,23 @@
  * If this structure is pointed to by a task_struct, next syscall to be called
  * by the task will have a non-default behavior.
  * For example, it can be used to pre-set the id of the object to be created
- * by next syscall.
+ * by next syscall. The following syscalls support this feature:
+ *   . msgget(), semget(), shmget()
  */
 struct next_syscall_data {
  int ndata;
  long data[NDATA];

```
 };

+/*
+ * Returns true if tsk has some data set in its next_syscall_data, 0 else
+ */
+#define next_data_set(tsk) ((tsk)->nsd    \
+     ? ((tsk)->nsd->ndata ? 1 : 0) \
+     : 0)
+
+#define get_next_data(tsk) ((tsk)->nsd->data[0])
+
 extern ssize_t get_next_syscall_data(struct task_struct *, char *, size_t);
 extern int set_next_syscall_data(struct task_struct *, char *);
 extern void reset_next_syscall_data(struct task_struct *);
Index: linux-2.6.26-rc8-mm1/ipc/util.c
===================================================================
--- linux-2.6.26-rc8-mm1.orig/ipc/util.c 2008-07-08 09:05:09.000000000 +0200
+++ linux-2.6.26-rc8-mm1/ipc/util.c 2008-07-08 12:13:40.000000000 +0200
@@ -266,20 +266,43 @@ int ipc_addid(struct ipc_ids* ids, struc
  if (ids->in_use >= size)
   return -ENOSPC;

- err = idr_get_new(&ids->ipcs_idr, new, &id);
- if (err)
-  return err;
+ if (unlikely(next_data_set(current))) {
+  /* There is a target id specified, try to use it */
+  int next_id = get_next_data(current);
+  int new_lid = next_id % SEQ_MULTIPLIER;
+  unsigned long new_seq = next_id / SEQ_MULTIPLIER;
+
+  reset_next_syscall_data(current);
+
+  if (next_id != (new_lid + (new_seq * SEQ_MULTIPLIER)))
+   return -EINVAL;
+
+  err = idr_get_new_above(&ids->ipcs_idr, new, new_lid, &id);
+  if (err)
+   return err;
+  if (id != new_lid) {
+   idr_remove(&ids->ipcs_idr, id);
+   return -EBUSY;
+  }
+
+  new->id = next_id;
+  new->seq = new_seq;
+ } else {
+  err = idr_get_new(&ids->ipcs_idr, new, &id);
```

```
+  if (err)
+   return err;
+
+  new->seq = ids->seq++;
+  if (ids->seq > ids->seq_max)
+   ids->seq = 0;
+  new->id = ipc_buildid(id, new->seq);
+ }

  ids->in_use++;

  new->cuid = new->uid = current->euid;
  new->gid = new->cgid = current->egid;

- new->seq = ids->seq++;
- if(ids->seq > ids->seq_max)
-  ids->seq = 0;
-
- new->id = ipc_buildid(id, new->seq);
  spin_lock_init(&new->lock);
  new->deleted = 0;
  rcu_read_lock();

--
```

Subject: [RFC PATCH 4/5] use next syscall data to change the behavior of IPC_SET

Posted by Nadia Derbey on Tue, 08 Jul 2008 11:24:26 GMT

View Forum Message <> Reply to Message

[PATCH 04/05]

This patch uses the value written into the next_syscall_data proc file
as a flag to change the way msgctl(IPC_SET), semctl(IPC_SET) and
shmctl(IPC_SET) behave.

When "LONG1 1" is echoed to this file, xxxctl(IPC_SET) will set the time
fields and the pid fields according to what is specified in the input
parameter (while currently only the permission fields are allowed to be set).
The following syscalls are impacted:
. msgctl(IPC_SET)
. semctl(IPC_SET)
. shmctl(IPC_SET)

This makes it easy to restart an ipc object exactly is it was during the
checkpoint phase.

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

---
 include/linux/next_syscall_data.h | 12 ++++++++++
 ipc/msg.c | 19 ++++++++++++++++++-
 ipc/sem.c | 16 ++++++++++++++-
 ipc/shm.c | 19 ++++++++++++++++++-
 4 files changed, 63 insertions(+), 3 deletions(-)

Index: linux-2.6.26-rc8-mm1/include/linux/next_syscall_data.h
===
--- linux-2.6.26-rc8-mm1.orig/include/linux/next_syscall_data.h 2008-07-08 12:22:47.000000000
+0200
+++ linux-2.6.26-rc8-mm1/include/linux/next_syscall_data.h 2008-07-08 12:24:29.000000000
+0200
@@ -6,6 +6,7 @@
  *   . object creation with a predefined id
  *   . for a sysv ipc object
  *   . for a process
+ *   . set more than the usual ipc_perm fields during and IPC_SET operation.
  */

 #ifndef _LINUX_NEXT_SYSCALL_DATA_H
@@ -20,6 +21,10 @@
  * by next syscall. The following syscalls support this feature:
  *   . msgget(), semget(), shmget()
  *   . fork(), vfork(), clone()
+ *
+ * If it is set to a non null value before a call to:
+ *   . msgctl(IPC_SET), semctl(IPC_SET), shmctl(IPC_SET),
+ * this means that we are going to set more than the usual ipc_perms fields.
  */
 struct next_syscall_data {
  int ndata;
@@ -35,6 +40,13 @@ struct next_syscall_data {

 #define get_next_data(tsk) ((tsk)->nsd->data[0])

+/*
+ * Returns true if next call to xxxctl(IPC_SET) should have a non-default
+ * behavior.
+ */
+#define ipc_set_all(tsk) (next_data_set(tsk) ? get_next_data(tsk) : 0)
+

```
+
 extern ssize_t get_next_syscall_data(struct task_struct *, char *, size_t);
 extern int set_next_syscall_data(struct task_struct *, char *);
 extern void reset_next_syscall_data(struct task_struct *);
Index: linux-2.6.26-rc8-mm1/ipc/msg.c
===================================================================
--- linux-2.6.26-rc8-mm1.orig/ipc/msg.c 2008-07-08 12:12:36.000000000 +0200
+++ linux-2.6.26-rc8-mm1/ipc/msg.c 2008-07-08 12:26:03.000000000 +0200
@@ -446,7 +446,24 @@ static int msgctl_down(struct ipc_namesp
   msq->q_qbytes = msqid64.msg_qbytes;

   ipc_update_perm(&msqid64.msg_perm, ipcp);
-  msq->q_ctime = get_seconds();
+  if (unlikely(ipc_set_all(current))) {
+   /*
+    * If this field is set in the task struct, this
+    * means that we want to set more than the usual
+    * fields. Particularly useful to restart a msgq
+    * in the same state as it was before being
+    * checkpointed.
+    */
+   msq->q_stime = msqid64.msg_stime;
+   msq->q_rtime = msqid64.msg_rtime;
+   msq->q_ctime = msqid64.msg_ctime;
+   msq->q_lspid = msqid64.msg_lspid;
+   msq->q_lrpid = msqid64.msg_lrpid;
+
+   reset_next_syscall_data(current);
+  } else
+   msq->q_ctime = get_seconds();
+
   /* sleeping receivers might be excluded by
    * stricter permissions.
    */
Index: linux-2.6.26-rc8-mm1/ipc/sem.c
===================================================================
--- linux-2.6.26-rc8-mm1.orig/ipc/sem.c 2008-07-08 12:12:36.000000000 +0200
+++ linux-2.6.26-rc8-mm1/ipc/sem.c 2008-07-08 12:27:06.000000000 +0200
@@ -874,7 +874,21 @@ static int semctl_down(struct ipc_namesp
   goto out_up;
  case IPC_SET:
   ipc_update_perm(&semid64.sem_perm, ipcp);
-  sma->sem_ctime = get_seconds();
+
+  if (unlikely(ipc_set_all(current))) {
+   /*
+    * If this field is set in the task struct, this
+    * means that we want to set more than the usual
```

```
+    * fields. Particularly useful to restart a semaphore
+    * in the same state as it was before being
+    * checkpointed.
+    */
+   sma->sem_ctime = semid64.sem_ctime;
+   sma->sem_otime = semid64.sem_otime;
+
+   reset_next_syscall_data(current);
+  } else
+   sma->sem_ctime = get_seconds();
    break;
   default:
    err = -EINVAL;
Index: linux-2.6.26-rc8-mm1/ipc/shm.c
===================================================================
--- linux-2.6.26-rc8-mm1.orig/ipc/shm.c 2008-07-08 12:12:36.000000000 +0200
+++ linux-2.6.26-rc8-mm1/ipc/shm.c 2008-07-08 12:27:32.000000000 +0200
@@ -609,7 +609,24 @@ static int shmctl_down(struct ipc_namesp
    goto out_up;
   case IPC_SET:
    ipc_update_perm(&shmid64.shm_perm, ipcp);
-   shp->shm_ctim = get_seconds();
+
+   if (unlikely(ipc_set_all(current))) {
+    /*
+     * If this field is set in the task struct, this
+     * means that we want to set more than the usual
+     * fields. Particularly useful to restart a shm seg
+     * in the same state as it was before being
+     * checkpointed.
+     */
+    shp->shm_atim = shmid64.shm_atime;
+    shp->shm_dtim = shmid64.shm_dtime;
+    shp->shm_ctim = shmid64.shm_ctime;
+    shp->shm_cprid = shmid64.shm_cpid;
+    shp->shm_lprid = shmid64.shm_lpid;
+
+    reset_next_syscall_data(current);
+   } else
+    shp->shm_ctim = get_seconds();
    break;
   default:
    err = -EINVAL;

--
```

Subject: Re: [RFC PATCH 1/5] adds the procfs facilities
Posted by serue on Tue, 08 Jul 2008 19:32:54 GMT
View Forum Message <> Reply to Message

Quoting Nadia.Derbey@bull.net (Nadia.Derbey@bull.net):
> [PATCH 01/05]
>
> This patch adds the procfs facility needed to feed some data for the
> next syscall to be called.
>
> The effect of issuing
> echo "LONG<Y> <XX>" > /proc/self/task/<tid>/next_syscall_data
> is that <XX> will be stored in a new field of the task structure
> (next_syscall_data). This field, in turn will be taken as the data to feed
> next syscall that supports the feature.
>
> <Y> is the number of values provided on the line.
> For the sake of simplicity it is now fixed to 1, but this can be extended as
> needed, in the future.
>
> This is particularly useful when restarting an application, as we need
> sometimes the syscalls to have a non-default behavior.
>
> Signed-off-by: Nadia Derbey <Nadia.Derbey@bull.net>

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

thanks,
-serge

>
> ---
> fs/exec.c | 6 +
> fs/proc/base.c | 75 ++++++++++++++++++++
> include/linux/next_syscall_data.h | 32 ++++++++
> include/linux/sched.h | 6 +
> kernel/Makefile | 3
> kernel/exit.c | 4 +
> kernel/fork.c | 2
> kernel/next_syscall_data.c | 151 ++
> 8 files changed, 278 insertions(+), 1 deletion(-)
>
> Index: linux-2.6.26-rc8-mm1/include/linux/sched.h
> ===
> --- linux-2.6.26-rc8-mm1.orig/include/linux/sched.h 2008-07-08 09:04:21.000000000 +0200

> +++ linux-2.6.26-rc8-mm1/include/linux/sched.h 2008-07-08 09:13:43.000000000 +0200
> @@ -87,6 +87,7 @@ struct sched_param {
> #include <linux/task_io_accounting.h>
> #include <linux/kobject.h>
> #include <linux/latencytop.h>
> +#include <linux/next_syscall_data.h>
>
> #include <asm/processor.h>
>
> @@ -1296,6 +1297,11 @@ struct task_struct {
>   int latency_record_count;
>   struct latency_record latency_record[LT_SAVECOUNT];
> #endif
> + /*
> + * If non-NULL indicates that next operation will be forced, e.g.
> + * that next object to be created will have a predefined id.
> + */
> + struct next_syscall_data *nsd;
> };
>
> /*
> Index: linux-2.6.26-rc8-mm1/include/linux/next_syscall_data.h
> ===
> --- /dev/null 1970-01-01 00:00:00.000000000 +0000
> +++ linux-2.6.26-rc8-mm1/include/linux/next_syscall_data.h 2008-07-08 09:24:38.000000000
+0200
> @@ -0,0 +1,32 @@
> +/*
> + * include/linux/next_syscall_data.h
> + *
> + * Definitions to support fixed data for next syscall to be called.
> + */
> +
> +#ifndef _LINUX_NEXT_SYSCALL_DATA_H
> +#define _LINUX_NEXT_SYSCALL_DATA_H
> +
> +#define NDATA 1
> +
> +/*
> + * If this structure is pointed to by a task_struct, next syscall to be called
> + * by the task will have a non-default behavior.
> + * For example, it can be used to pre-set the id of the object to be created
> + * by next syscall.
> + */
> +struct next_syscall_data {
> + int ndata;
> + long data[NDATA];
> +};

```
> +
> +extern ssize_t get_next_syscall_data(struct task_struct *, char *, size_t);
> +extern int set_next_syscall_data(struct task_struct *, char *);
> +extern void reset_next_syscall_data(struct task_struct *);
> +
> +static inline void exit_next_syscall_data(struct task_struct *tsk)
> +{
> + reset_next_syscall_data(tsk);
> +}
> +
> +#endif /* _LINUX_NEXT_SYSCALL_DATA_H */
> Index: linux-2.6.26-rc8-mm1/fs/proc/base.c
> ===================================================================
> --- linux-2.6.26-rc8-mm1.orig/fs/proc/base.c 2008-07-08 09:05:13.000000000 +0200
> +++ linux-2.6.26-rc8-mm1/fs/proc/base.c 2008-07-08 09:18:12.000000000 +0200
> @@ -1158,6 +1158,76 @@ static const struct file_operations proc
>  };
>  #endif
>
> +static ssize_t next_syscall_data_read(struct file *file, char __user *buf,
> +    size_t count, loff_t *ppos)
> +{
> + struct task_struct *task;
> + char *page;
> + ssize_t length;
> +
> + task = get_proc_task(file->f_path.dentry->d_inode);
> + if (!task)
> +  return -ESRCH;
> +
> + if (count >= PAGE_SIZE)
> +  count = PAGE_SIZE - 1;
> +
> + length = -ENOMEM;
> + page = (char *) __get_free_page(GFP_TEMPORARY);
> + if (!page)
> +  goto out;
> +
> + length = get_next_syscall_data(task, (char *) page, count);
> + if (length >= 0)
> +  length = simple_read_from_buffer(buf, count, ppos,
> +      (char *)page, length);
> + free_page((unsigned long) page);
> +
> +out:
> + put_task_struct(task);
> + return length;
> +}
```

```
> +
> +static ssize_t next_syscall_data_write(struct file *file,
> +    const char __user *buf,
> +    size_t count, loff_t *ppos)
> +{
> + struct inode *inode = file->f_path.dentry->d_inode;
> + char *page;
> + ssize_t length;
> +
> + if (pid_task(proc_pid(inode), PIDTYPE_PID) != current)
> +  return -EPERM;
> +
> + if (count >= PAGE_SIZE)
> +  count = PAGE_SIZE - 1;
> +
> + if (*ppos != 0) {
> +  /* No partial writes. */
> +  return -EINVAL;
> + }
> + page = (char *)__get_free_page(GFP_TEMPORARY);
> + if (!page)
> +  return -ENOMEM;
> + length = -EFAULT;
> + if (copy_from_user(page, buf, count))
> +  goto out_free_page;
> +
> + page[count] = '\0';
> +
> + length = set_next_syscall_data(current, page);
> + if (!length)
> +  length = count;
> +
> +out_free_page:
> + free_page((unsigned long) page);
> + return length;
> +}
> +
> +static const struct file_operations proc_next_syscall_data_operations = {
> + .read  = next_syscall_data_read,
> + .write  = next_syscall_data_write,
> +};
>
>  #ifdef CONFIG_SCHED_DEBUG
>  /*
> @@ -2853,6 +2923,11 @@ static const struct pid_entry tid_base_s
>  #ifdef CONFIG_TASK_IO_ACCOUNTING
>   INF("io", S_IRUGO, tid_io_accounting),
>  #endif
```

> + /*
> + * NOTE that this file is not added into tgid_base_stuff[] since it
> + * has to be specified on a per-thread basis.
> + */
> + REG("next_syscall_data", S_IRUGO|S_IWUSR, next_syscall_data),
> };
>
> static int proc_tid_base_readdir(struct file * filp,
> Index: linux-2.6.26-rc8-mm1/kernel/Makefile
> ===
> --- linux-2.6.26-rc8-mm1.orig/kernel/Makefile 2008-07-08 09:04:35.000000000 +0200
> +++ linux-2.6.26-rc8-mm1/kernel/Makefile 2008-07-08 09:19:14.000000000 +0200
> @@ -9,7 +9,8 @@ obj-y   = sched.o fork.o exec_domain.o
>   rcupdate.o extable.o params.o posix-timers.o \
>   kthread.o wait.o kfifo.o sys_ni.o posix-cpu-timers.o mutex.o \
>   hrtimer.o rwsem.o nsproxy.o srcu.o semaphore.o \
> -   notifier.o ksysfs.o pm_qos_params.o sched_clock.o
> +   notifier.o ksysfs.o pm_qos_params.o sched_clock.o \
> +   next_syscall_data.o
>
> CFLAGS_REMOVE_sched.o = -pg -mno-spe
>
> Index: linux-2.6.26-rc8-mm1/kernel/next_syscall_data.c
> ===
> --- /dev/null 1970-01-01 00:00:00.000000000 +0000
> +++ linux-2.6.26-rc8-mm1/kernel/next_syscall_data.c 2008-07-08 09:35:27.000000000 +0200
> @@ -0,0 +1,151 @@
> +/*
> + * linux/kernel/next_syscall_data.c
> + *
> + *
> + * Provide the get_next_syscall_data() / set_next_syscall_data() routines
> + * (called from fs/proc/base.c).
> + * They allow to specify some particular data for the next syscall to be
> + * called.
> + * E.g. they can be used to specify the id for the next resource to be
> + * allocated, instead of letting the allocator set it for us.
> + */
> +
> +#include <linux/sched.h>
> +#include <linux/ctype.h>
> +
> +
> +
> +ssize_t get_next_syscall_data(struct task_struct *task, char *buffer,
> +   size_t size)
> +{
> + struct next_syscall_data *nsd;

```
> + char *bufptr = buffer;
> + ssize_t rc, count = 0;
> + int i;
> +
> + nsd = task->nsd;
> + if (!nsd || !nsd->ndata)
> +  return snprintf(buffer, size, "UNSET\n");
> +
> + count = snprintf(bufptr, size, "LONG%d ", nsd->ndata);
> +
> + for (i = 0; i < nsd->ndata - 1; i++) {
> +  rc = snprintf(&bufptr[count], size - count, "%ld ",
> +    nsd->data[i]);
> +  if (rc >= size - count)
> +   return -ENOMEM;
> +  count += rc;
> + }
> +
> + rc = snprintf(&bufptr[count], size - count, "%ld\n", nsd->data[i]);
> + if (rc >= size - count)
> +  return -ENOMEM;
> + count += rc;
> +
> + return count;
> +}
> +
> +static int fill_next_syscall_data(struct task_struct *task, int ndata,
> +    char *buffer)
> +{
> + char *token, *buff = buffer;
> + char *end;
> + struct next_syscall_data *nsd = task->nsd;
> + int i;
> +
> + if (!nsd) {
> +  nsd = kmalloc(sizeof(*nsd), GFP_KERNEL);
> +  if (!nsd)
> +   return -ENOMEM;
> +  task->nsd = nsd;
> + }
> +
> + nsd->ndata = ndata;
> +
> + i = 0;
> + while ((token = strsep(&buff, " ")) != NULL && i < ndata) {
> +  long data;
> +
> +  if (!*token)
```

```
> +   goto out_free;
> +  data = simple_strtol(token, &end, 0);
> +  if (end == token || (*end && !isspace(*end)))
> +   goto out_free;
> +  nsd->data[i] = data;
> +  i++;
> + }
> +
> + if (i != ndata)
> +  goto out_free;
> +
> + return 0;
> +
> +out_free:
> + kfree(nsd);
> + task->nsd = NULL;
> + return -EINVAL;
> +}
> +
> +/*
> + * Parses a line with the following format:
> + * <x> <id0> ... <idx-1>
> + * Currently, only x=1 is accepted.
> + * Any trailing character on the line is skipped.
> + */
> +static int do_set_next_syscall_data(struct task_struct *task, char *nb,
> +     char *buffer)
> +{
> + int ndata;
> + char *end;
> +
> + ndata = simple_strtol(nb, &end, 0);
> + if (*end)
> +  return -EINVAL;
> +
> + if (ndata > NDATA)
> +  return -EINVAL;
> +
> + return fill_next_syscall_data(task, ndata, buffer);
> +}
> +
> +void reset_next_syscall_data(struct task_struct *task)
> +{
> + struct next_syscall_data *nsd = task->nsd;
> +
> + if (nsd) {
> +  task->nsd = NULL;
> +  kfree(nsd);
```

```
> + }
> +}
> +
> +#define LONG_STR "LONG"
> +#define RESET_STR "RESET"
> +
> +/*
> + * Parses a line written to /proc/self/task/<my_tid>/next_syscall_data.
> + * this line has the following format:
> + * LONG<x> id              --> a sequence of id(s) is specified
> + *                           currently, only x=1 is accepted
> + */
> +int set_next_syscall_data(struct task_struct *task, char *buffer)
> +{
> + char *token, *out = buffer;
> + size_t sz;
> +
> + if (!out)
> +  return -EINVAL;
> +
> + token = strsep(&out, " ");
> +
> + sz = strlen(LONG_STR);
> +
> + if (!strncmp(token, LONG_STR, sz))
> +  return do_set_next_syscall_data(task, token + sz, out);
> +
> + if (!strncmp(token, RESET_STR, strlen(RESET_STR))) {
> +  reset_next_syscall_data(task);
> +  return 0;
> + }
> +
> + return -EINVAL;
> +}
> Index: linux-2.6.26-rc8-mm1/kernel/fork.c
> ===================================================================
> --- linux-2.6.26-rc8-mm1.orig/kernel/fork.c 2008-07-08 09:04:35.000000000 +0200
> +++ linux-2.6.26-rc8-mm1/kernel/fork.c 2008-07-08 09:25:35.000000000 +0200
> @@ -1085,6 +1085,8 @@ static struct task_struct *copy_process(
>   p->blocked_on = NULL; /* not blocked yet */
>  #endif
>
> + p->nsd = NULL; /* no next syscall data is the default */
> +
>   /* Perform scheduler related setup. Assign this task to a CPU. */
>   sched_fork(p, clone_flags);
>
> Index: linux-2.6.26-rc8-mm1/fs/exec.c
```

> ===
> --- linux-2.6.26-rc8-mm1.orig/fs/exec.c 2008-07-08 09:05:13.000000000 +0200
> +++ linux-2.6.26-rc8-mm1/fs/exec.c 2008-07-08 09:26:21.000000000 +0200
> @@ -1016,6 +1016,12 @@ int flush_old_exec(struct linux_binprm *
> flush_signal_handlers(current, 0);
> flush_old_files(current->files);
>
> + /*
> + * the next syscall data is not inherited across execve()
> + */
> + if (unlikely(current->nsd))
> + reset_next_syscall_data(current);
> +
> return 0;
>
> out:
> Index: linux-2.6.26-rc8-mm1/kernel/exit.c
> ===
> --- linux-2.6.26-rc8-mm1.orig/kernel/exit.c 2008-07-08 09:04:35.000000000 +0200
> +++ linux-2.6.26-rc8-mm1/kernel/exit.c 2008-07-08 09:27:31.000000000 +0200
> @@ -1066,6 +1066,10 @@ NORET_TYPE void do_exit(long code)
>
> proc_exit_connector(tsk);
> exit_notify(tsk, group_dead);
> +
> + if (unlikely(tsk->nsd))
> + exit_next_syscall_data(tsk);
> +
> #ifdef CONFIG_NUMA
> mpol_put(tsk->mempolicy);
> tsk->mempolicy = NULL;
>
> --

_____

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

---

Subject: Re: [RFC PATCH 2/5] use next syscall data to predefine ipc objects ids
Posted by serue on Tue, 08 Jul 2008 19:38:08 GMT
View Forum Message <> Reply to Message

Quoting Nadia.Derbey@bull.net (Nadia.Derbey@bull.net):
> [PATCH 02/05]
>
> This patch uses the value written into the next_syscall_data proc file
> as a target id for the next IPC object to be created.

> The following syscalls have a new behavior if next_syscall_data is set:
> . mssget()
> . semget()
> . shmget()
>
> Signed-off-by: Nadia Derbey <Nadia.Derbey@bull.net>

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

thanks,
-serge

>
> ---
> include/linux/next_syscall_data.h | 17 +++++++++++++--
> ipc/util.c | 39 +++++++++++++++++++++++++++++++++++++++--------
> 2 files changed, 46 insertions(+), 10 deletions(-)
>
> Index: linux-2.6.26-rc8-mm1/include/linux/next_syscall_data.h
> ===
> --- linux-2.6.26-rc8-mm1.orig/include/linux/next_syscall_data.h 2008-07-08 09:24:38.000000000
+0200
> +++ linux-2.6.26-rc8-mm1/include/linux/next_syscall_data.h 2008-07-08 12:12:39.000000000
+0200
> @@ -1,7 +1,10 @@
> /*
>   * include/linux/next_syscall_data.h
>   *
> - * Definitions to support fixed data for next syscall to be called.
> + * Definitions to support fixed data for next syscall to be called. The
> + * following is supported today:
> + *   . object creation with a predefined id
> + *   . for a sysv ipc object
>   */
>
> #ifndef _LINUX_NEXT_SYSCALL_DATA_H
> @@ -13,13 +16,23 @@
>   * If this structure is pointed to by a task_struct, next syscall to be called
>   * by the task will have a non-default behavior.
>   * For example, it can be used to pre-set the id of the object to be created
> - * by next syscall.
> + * by next syscall. The following syscalls support this feature:
> + *   . msgget(), semget(), shmget()
>   */
> struct next_syscall_data {
>   int ndata;
>   long data[NDATA];
> };

>
> +/*
> + * Returns true if tsk has some data set in its next_syscall_data, 0 else
> + */
> +#define next_data_set(tsk) ((tsk)->nsd   \
> +   ? ((tsk)->nsd->ndata ? 1 : 0) \
> +   : 0)
> +
> +#define get_next_data(tsk) ((tsk)->nsd->data[0])
> +
> extern ssize_t get_next_syscall_data(struct task_struct *, char *, size_t);
> extern int set_next_syscall_data(struct task_struct *, char *);
> extern void reset_next_syscall_data(struct task_struct *);
> Index: linux-2.6.26-rc8-mm1/ipc/util.c
> ===
> --- linux-2.6.26-rc8-mm1.orig/ipc/util.c 2008-07-08 09:05:09.000000000 +0200
> +++ linux-2.6.26-rc8-mm1/ipc/util.c 2008-07-08 12:13:40.000000000 +0200
> @@ -266,20 +266,43 @@ int ipc_addid(struct ipc_ids* ids, struc
>   if (ids->in_use >= size)
>   return -ENOSPC;
>
> - err = idr_get_new(&ids->ipcs_idr, new, &id);
> - if (err)
> - return err;
> + if (unlikely(next_data_set(current))) {
> + /* There is a target id specified, try to use it */
> + int next_id = get_next_data(current);
> + int new_lid = next_id % SEQ_MULTIPLIER;
> + unsigned long new_seq = next_id / SEQ_MULTIPLIER;
> +
> + reset_next_syscall_data(current);
> +
> + if (next_id != (new_lid + (new_seq * SEQ_MULTIPLIER)))
> +   return -EINVAL;
> +
> + err = idr_get_new_above(&ids->ipcs_idr, new, new_lid, &id);
> + if (err)
> +   return err;
> + if (id != new_lid) {
> +   idr_remove(&ids->ipcs_idr, id);
> +   return -EBUSY;
> + }
> +
> + new->id = next_id;
> + new->seq = new_seq;
> + } else {
> + err = idr_get_new(&ids->ipcs_idr, new, &id);
> + if (err)

> +   return err;
> +
> + new->seq = ids->seq++;
> + if (ids->seq > ids->seq_max)
> +   ids->seq = 0;
> + new->id = ipc_buildid(id, new->seq);
> + }
>
>   ids->in_use++;
>
>   new->cuid = new->uid = current->euid;
>   new->gid = new->cgid = current->egid;
>
> - new->seq = ids->seq++;
> - if(ids->seq > ids->seq_max)
> - ids->seq = 0;
> -
> - new->id = ipc_buildid(id, new->seq);
>   spin_lock_init(&new->lock);
>   new->deleted = 0;
>   rcu_read_lock();
>
> --

Subject: Re: [RFC PATCH 4/5] use next syscall data to change the behavior of IPC_SET

Posted by serue on Tue, 08 Jul 2008 19:56:18 GMT

View Forum Message <> Reply to Message

Quoting Nadia.Derbey@bull.net (Nadia.Derbey@bull.net):
> [PATCH 04/05]
>
> This patch uses the value written into the next_syscall_data proc file
> as a flag to change the way msgctl(IPC_SET), semctl(IPC_SET) and
> shmctl(IPC_SET) behave.
>
> When "LONG1 1" is echoed to this file, xxxctl(IPC_SET) will set the time
> fields and the pid fields according to what is specified in the input
> parameter (while currently only the permission fields are allowed to be set).
> The following syscalls are impacted:
> . msgctl(IPC_SET)
> . semctl(IPC_SET)
> . shmctl(IPC_SET)

>
> This makes it easy to restart an ipc object exactly is it was during the
> checkpoint phase.
>
> Signed-off-by: Nadia Derbey <Nadia.Derbey@bull.net>

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

thanks,
-serge

> ---
> include/linux/next_syscall_data.h | 12 ++++++++++
> ipc/msg.c | 19 ++++++++++++++++++-
> ipc/sem.c | 16 ++++++++++++++-
> ipc/shm.c | 19 ++++++++++++++++++-
> 4 files changed, 63 insertions(+), 3 deletions(-)
>
> Index: linux-2.6.26-rc8-mm1/include/linux/next_syscall_data.h
> ===
> --- linux-2.6.26-rc8-mm1.orig/include/linux/next_syscall_data.h 2008-07-08 12:22:47.000000000
+0200
> +++ linux-2.6.26-rc8-mm1/include/linux/next_syscall_data.h 2008-07-08 12:24:29.000000000
+0200
> @@ -6,6 +6,7 @@
>   *   . object creation with a predefined id
>   *   . for a sysv ipc object
>   *   . for a process
> + *   . set more than the usual ipc_perm fields during and IPC_SET operation.
>   */
>
> #ifndef _LINUX_NEXT_SYSCALL_DATA_H
> @@ -20,6 +21,10 @@
>   * by next syscall. The following syscalls support this feature:
>   *   . msgget(), semget(), shmget()
>   *   . fork(), vfork(), clone()
> + *
> + * If it is set to a non null value before a call to:
> + *   . msgctl(IPC_SET), semctl(IPC_SET), shmctl(IPC_SET),
> + * this means that we are going to set more than the usual ipc_perms fields.
>   */
> struct next_syscall_data {
>   int ndata;
> @@ -35,6 +40,13 @@ struct next_syscall_data {
>
> #define get_next_data(tsk) ((tsk)->nsd->data[0])
>
> +/*

```
> + * Returns true if next call to xxxctl(IPC_SET) should have a non-default
> + * behavior.
> + */
> +#define ipc_set_all(tsk) (next_data_set(tsk) ? get_next_data(tsk) : 0)
> +
> +
>  extern ssize_t get_next_syscall_data(struct task_struct *, char *, size_t);
>  extern int set_next_syscall_data(struct task_struct *, char *);
>  extern void reset_next_syscall_data(struct task_struct *);
> Index: linux-2.6.26-rc8-mm1/ipc/msg.c
> ===================================================================
> --- linux-2.6.26-rc8-mm1.orig/ipc/msg.c 2008-07-08 12:12:36.000000000 +0200
> +++ linux-2.6.26-rc8-mm1/ipc/msg.c 2008-07-08 12:26:03.000000000 +0200
> @@ -446,7 +446,24 @@ static int msgctl_down(struct ipc_namesp
>    msq->q_qbytes = msqid64.msg_qbytes;
>
>    ipc_update_perm(&msqid64.msg_perm, ipcp);
> -  msq->q_ctime = get_seconds();
> +  if (unlikely(ipc_set_all(current))) {
> +   /*
> +    * If this field is set in the task struct, this
> +    * means that we want to set more than the usual
> +    * fields. Particularly useful to restart a msgq
> +    * in the same state as it was before being
> +    * checkpointed.
> +    */
> +   msq->q_stime = msqid64.msg_stime;
> +   msq->q_rtime = msqid64.msg_rtime;
> +   msq->q_ctime = msqid64.msg_ctime;
> +   msq->q_lspid = msqid64.msg_lspid;
> +   msq->q_lrpid = msqid64.msg_lrpid;
> +
> +   reset_next_syscall_data(current);
> +  } else
> +   msq->q_ctime = get_seconds();
> +
>    /* sleeping receivers might be excluded by
>     * stricter permissions.
>     */
> Index: linux-2.6.26-rc8-mm1/ipc/sem.c
> ===================================================================
> --- linux-2.6.26-rc8-mm1.orig/ipc/sem.c 2008-07-08 12:12:36.000000000 +0200
> +++ linux-2.6.26-rc8-mm1/ipc/sem.c 2008-07-08 12:27:06.000000000 +0200
> @@ -874,7 +874,21 @@ static int semctl_down(struct ipc_namesp
>    goto out_up;
>   case IPC_SET:
>    ipc_update_perm(&semid64.sem_perm, ipcp);
> -  sma->sem_ctime = get_seconds();
```

```
> +
> +  if (unlikely(ipc_set_all(current))) {
> +   /*
> +    * If this field is set in the task struct, this
> +    * means that we want to set more than the usual
> +    * fields. Particularly useful to restart a semaphore
> +    * in the same state as it was before being
> +    * checkpointed.
> +    */
> +   sma->sem_ctime = semid64.sem_ctime;
> +   sma->sem_otime = semid64.sem_otime;
> +
> +   reset_next_syscall_data(current);
> +  } else
> +   sma->sem_ctime = get_seconds();
>    break;
>   default:
>    err = -EINVAL;
> Index: linux-2.6.26-rc8-mm1/ipc/shm.c
> ===================================================================
> --- linux-2.6.26-rc8-mm1.orig/ipc/shm.c 2008-07-08 12:12:36.000000000 +0200
> +++ linux-2.6.26-rc8-mm1/ipc/shm.c 2008-07-08 12:27:32.000000000 +0200
> @@ -609,7 +609,24 @@ static int shmctl_down(struct ipc_namesp
>    goto out_up;
>   case IPC_SET:
>    ipc_update_perm(&shmid64.shm_perm, ipcp);
> -  shp->shm_ctim = get_seconds();
> +
> +  if (unlikely(ipc_set_all(current))) {
> +   /*
> +    * If this field is set in the task struct, this
> +    * means that we want to set more than the usual
> +    * fields. Particularly useful to restart a shm seg
> +    * in the same state as it was before being
> +    * checkpointed.
> +    */
> +   shp->shm_atim = shmid64.shm_atime;
> +   shp->shm_dtim = shmid64.shm_dtime;
> +   shp->shm_ctim = shmid64.shm_ctime;
> +   shp->shm_cprid = shmid64.shm_cpid;
> +   shp->shm_lprid = shmid64.shm_lpid;
> +
> +   reset_next_syscall_data(current);
> +  } else
> +   shp->shm_ctim = get_seconds();
>    break;
>   default:
>    err = -EINVAL;
```

>
> --

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

---

Subject: Re: [RFC PATCH 0/5] Resend -v2 - Use procfs to change a syscall behavior
Posted by Alexey Dobriyan on Wed, 09 Jul 2008 22:10:28 GMT
View Forum Message <> Reply to Message

On Tue, Jul 08, 2008 at 01:24:22PM +0200, Nadia.Derbey@bull.net wrote:
> # echo "LONG1 XX" > /proc/self/task/<my_tid>/next_syscall_data

Same stuff.

There is struct task_struct::did_exec , what about it?

Also, patches are about de-serializing, how serializing from userspace looks like?
You freezed group of processes, then what?

How, for example, dump all VMAs correctly?
[prepares counter-example]

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

---

Subject: Re: [RFC PATCH 0/5] Resend -v2 - Use procfs to change a syscall behavior
Posted by ebiederm on Thu, 10 Jul 2008 00:36:46 GMT
View Forum Message <> Reply to Message

Nadia.Derbey@bull.net writes:

> This patchset is a part of an effort to change some syscalls behavior for
> checkpoint restart.

Thanks for doing this.

Unfortunately this makes a very good case of why we don't want to go down
this route. Adding magic parameters to syscalls that are only useful
in one very specific restart case.

We need good clean interfaces with well defined semantics.

Something as narrow focused on this is not really useful and it takes
a lot of code to do something very few people will want to actively
do.

> The syntax is:
> # echo "LONG1 XX" > /proc/self/task/<my_tid>/next_syscall_data
> next object to be created will have an id set to XX

Which his horrible in another way because it is hugely race prone.

Eric

Subject: Re: [RFC PATCH 0/5] Resend -v2 - Use procfs to change a syscall behavior
Posted by ebiederm on Thu, 10 Jul 2008 00:43:04 GMT
View Forum Message <> Reply to Message

Alexey Dobriyan <adobriyan@gmail.com> writes:

> On Tue, Jul 08, 2008 at 01:24:22PM +0200, Nadia.Derbey@bull.net wrote:
>> # echo "LONG1 XX" > /proc/self/task/<my_tid>/next_syscall_data
>
> Same stuff.
>
> There is struct task_struct::did_exec , what about it?
>
> Also, patches are about de-serializing, how serializing from userspace looks
> like?
> You freezed group of processes, then what?
>
> How, for example, dump all VMAs correctly?
> [prepares counter-example]

Alexey userspace vs a kernel space implementation is the wrong argument.

It is clearly established that the current user space interfaces are
insufficient to do the job. So we need to implement something in the kernel.

Further I have heard of no one suggesting running a single kernel on multiple

machines. Therefore there no one seems to be doing this entirely in the kernel
and so we need a user space component.

So the question should not be user space vs. kernel space but can we build clean
interfaces for checkpoint/restart? What will those interfaces be?

Although I think it is good that we are seeing more people play with this as
that should mean that our pool of people for doing code review on the implementation
should be reasonable.

Eric

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

---

Subject: Re: [RFC PATCH 0/5] Resend -v2 - Use procfs to change a syscall behavior
Posted by Alexey Dobriyan on Thu, 10 Jul 2008 01:39:15 GMT
View Forum Message <> Reply to Message

On Wed, Jul 09, 2008 at 05:43:04PM -0700, Eric W. Biederman wrote:
> Alexey Dobriyan <adobriyan@gmail.com> writes:
>
> > On Tue, Jul 08, 2008 at 01:24:22PM +0200, Nadia.Derbey@bull.net wrote:
> >> # echo "LONG1 XX" > /proc/self/task/<my_tid>/next_syscall_data
> >
> > Same stuff.
> >
> > There is struct task_struct::did_exec , what about it?
> >
> > Also, patches are about de-serializing, how serializing from userspace looks
> > like?
> > You freezed group of processes, then what?
> >
> > How, for example, dump all VMAs correctly?
> > [prepares counter-example]
>
> Alexey userspace vs a kernel space implementation is the wrong argument.
>
> It is clearly established that the current user space interfaces are
> insufficient to do the job. So we need to implement something in the kernel.
>
> Further I have heard of no one suggesting running a single kernel on multiple
> machines. Therefore there no one seems to be doing this entirely in the kernel
> and so we need a user space component.
>

> So the question should not be user space vs. kernel space but can we build clean
> interfaces for checkpoint/restart?

> What will those interfaces be?

In case of ->did_exec the only clean interface I see is:

 tsk->did_exec = !!tsk_img->did_exec;

It would be pretty silly to wrap this one line in a system call (two
actually -- one in, one out), since you're going to restore some more
fields of such variety anyway (like ->pdeath_signal).

Given the diversity of kernel internal data structures and all sorts of
links between them, the only system call suitable is ioctl(2), not all
this zoo of system calls proposed. They are all extendable and without
rules, but ioctl(2) is also without rules.

This is all said in assumption that serializing kernel-internal data for
checkpoint/restart to userspace is acceptable for mainline.
I don't think it is.

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

---

Subject: Re: [RFC PATCH 0/5] Resend -v2 - Use procfs to change a syscall behavior
Posted by ebiederm on Thu, 10 Jul 2008 02:14:37 GMT
View Forum Message <> Reply to Message

Alexey Dobriyan <adobriyan@gmail.com> writes:

> In case of ->did_exec the only clean interface I see is:
>
> tsk->did_exec = !!tsk_img->did_exec;
>
> It would be pretty silly to wrap this one line in a system call (two
> actually -- one in, one out), since you're going to restore some more
> fields of such variety anyway (like ->pdeath_signal).

There I agree the granularity seems small enough to be a major pain
for the implementation.

> Given the diversity of kernel internal data structures and all sorts of

> links between them, the only system call suitable is ioctl(2), not all
> this zoo of system calls proposed. They are all extendable and without
> rules, but ioctl(2) is also without rules.

At least for processes my gut reaction is to look at binary formats
and coredumps. Something with at least that large of a granularity seems
to make most sense.

> This is all said in assumption that serializing kernel-internal data for
> checkpoint/restart to userspace is acceptable for mainline.
> I don't think it is.

I don't believe that serializing kernel-internal data is acceptable
for mainline. I believe that serializing user-visible data is acceptable.
Note: user-visible data does not mean user-manipulatable data.

On a socket you may not save the skbs but you can save the pending packets
for example. Assuming the transition cost is not too high.

Eric

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

---

Subject: Re: [RFC PATCH 0/5] Resend -v2 - Use procfs to change a syscall behavior
Posted by Nadia Derbey on Thu, 10 Jul 2008 09:54:07 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Nadia.Derbey@bull.net writes:
>
>
>>This patchset is a part of an effort to change some syscalls behavior for
>>checkpoint restart.
>
>
> Thanks for doing this.
>
> Unfortunately this makes a very good case of why we don't want to go down
> this route. Adding magic parameters to syscalls that are only useful
> in one very specific restart case.
>
> We need good clean interfaces with well defined semantics.
>
> Something as narrow focused on this is not really useful and it takes

> a lot of code to do something very few people will want to actively
> do.

All this seems reasonable.
Ok, so since we are taking the "new syscalls" direction, I'll try to
make a list of the potentially duplicated syscalls.

Regards,
Nadia

>
>
>>The syntax is:
>># echo "LONG1 XX" > /proc/self/task/<my_tid>/next_syscall_data
>>   next object to be created will have an id set to XX
>
>
> Which his horrible in another way because it is hugely race prone.
>
> Eric
>
>
>

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

---

Subject: Re: [RFC PATCH 0/5] Resend -v2 - Use procfs to change a syscall behavior
Posted by Dave Hansen on Thu, 10 Jul 2008 16:01:42 GMT
View Forum Message <> Reply to Message

On Thu, 2008-07-10 at 02:10 +0400, Alexey Dobriyan wrote:
> How, for example, dump all VMAs correctly?
> [prepares counter-example]

Are there some particular pitfalls that you'd like to share? I'd love
to hear some of the issues the you've run into with Virtuozzo as its
implementation was created.

-- Dave

_____

Subject: Re: [RFC PATCH 0/5] Resend -v2 - Use procfs to change a syscall behavior
Posted by ebiederm on Tue, 15 Jul 2008 18:18:29 GMT
View Forum Message <> Reply to Message

Alexey Dobriyan <adobriyan@gmail.com> writes:

> This is all said in assumption that serializing kernel-internal data for
> checkpoint/restart to userspace is acceptable for mainline.
> I don't think it is.

Just a quick comment here. We mentioned checkpoint/restart is where we were going last kernel summit, and no one was opposed.

So while I expect technical objects if we are not careful, I believe a well chosen checkpoint/restart framework has every chance of being merged into mainline.

Eric

Subject: Re: [RFC PATCH 0/5] Resend -v2 - Use procfs to change a syscall behavior
Posted by Oren Laadan on Thu, 17 Jul 2008 22:42:56 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Alexey Dobriyan <adobriyan@gmail.com> writes:

I seem to not have received any of Alexey's emails... ?

>
>> On Tue, Jul 08, 2008 at 01:24:22PM +0200, Nadia.Derbey@bull.net wrote:
>>> # echo "LONG1 XX" > /proc/self/task/<my_tid>/next_syscall_data
>> Same stuff.
>>
>> There is struct task_struct::did_exec , what about it?
>>

>> Also, patches are about de-serializing, how serializing from userspace looks
>> like?
>> You freezed group of processes, then what?
>>
>> How, for example, dump all VMAs correctly?
>> [prepares counter-example]
>
> Alexey userspace vs a kernel space implementation is the wrong argument.
>
> It is clearly established that the current user space interfaces are
> insufficient to do the job. So we need to implement something in the kernel.
>
> Further I have heard of no one suggesting running a single kernel on multiple
> machines. Therefore there no one seems to be doing this entirely in the kernel
> and so we need a user space component.

I'm not sure I understand this argument ?

In a kernel implementation, the component will merely open a file descriptor
(to which the data will be streamed), freeze the container and invoke a
system call. In a userland implementation, the component will do most of
the work by continuously probing the kernel for information about the
processes that are being checkpointed.

So, of course we need a "component" - but what does that component do ?

> So the question should not be user space vs. kernel space but can we build clean
> interfaces for checkpoint/restart? What will those interfaces be?

My question is why build a set of interfaces to export this and that from
the kernel to user space ? if a kernel implementation (with minimal user
space support) is chosen, then information extraction (and restoration) is
straightforward and we don't get ourselves tied until the end of times to
API exported to userland.

The output of the module will be a binary (like a core dump) that can be
used by the same module to restart. User utilities will be available to
inspect the contents (for whatever reason - like a debugger can inspect a
core dump), and moreover to convert between old and new formats when moving
from older to newer kernels.

By doing so, we avoid many API issues - design, complexity, contents, and
the amount of interfaces to be added.

By doing so, we also gain much in terms of atomicity, possibility to add
optimizations and improve performance, as well as add features as we wish,
without the burden of commitments to userspace.

I think the kernel space vs. user space must be the first issue on our
table to solve, as it has a wide impact on the rest of the work.

Oren.

>
> Although I think it is good that we are seeing more people play with this as
> that should mean that our pool of people for doing code review on the implementation
> should be reasonable.
>
> Eric
> _____
> 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

---

Subject: Re: [RFC PATCH 0/5] Resend -v2 - Use procfs to change a syscall behavior
Posted by Matt Helsley on Fri, 18 Jul 2008 01:09:25 GMT
View Forum Message <> Reply to Message

On Thu, 2008-07-17 at 18:42 -0400, Oren Laadan wrote:
>
> Eric W. Biederman wrote:
> > Alexey Dobriyan <adobriyan@gmail.com> writes:
>
> I seem to not have received any of Alexey's emails... ?
>
> >
> >> On Tue, Jul 08, 2008 at 01:24:22PM +0200, Nadia.Derbey@bull.net wrote:
> >>> # echo "LONG1 XX" > /proc/self/task/<my_tid>/next_syscall_data
> >> Same stuff.
> >>
> >> There is struct task_struct::did_exec , what about it?
> >>
> >> Also, patches are about de-serializing, how serializing from userspace looks
> >> like?
> >> You freezed group of processes, then what?
> >>
> >> How, for example, dump all VMAs correctly?
> >> [prepares counter-example]
> >
> > Alexey userspace vs a kernel space implementation is the wrong argument.

> >
> > It is clearly established that the current user space interfaces are
> > insufficient to do the job. So we need to implement something in the kernel.
> >
> > Further I have heard of no one suggesting running a single kernel on multiple
> > machines. Therefore there no one seems to be doing this entirely in the kernel
> > and so we need a user space component.
>
> I'm not sure I understand this argument ?
>
> In a kernel implementation, the component will merely open a file descriptor
> (to which the data will be streamed), freeze the container and invoke a
> system call. In a userland implementation, the component will do most of
> the work by continuously probing the kernel for information about the
> processes that are being checkpointed.
>
> So, of course we need a "component" - but what does that component do ?
>
> > So the question should not be user space vs. kernel space but can we build clean
> > interfaces for checkpoint/restart? What will those interfaces be?
>
> My question is why build a set of interfaces to export this and that from
> the kernel to user space ? if a kernel implementation (with minimal user
> space support) is chosen, then information extraction (and restoration) is
> straightforward and we don't get ourselves tied until the end of times to
> API exported to userland.

That still seems like an API exported to userland. It just combines the
data into one block rather than distributing it amongst a bunch of
pseudo-filesystems. Does this form of API really free us from always
supporting it in the future?

> The output of the module will be a binary (like a core dump) that can be
> used by the same module to restart. User utilities will be available to
> inspect the contents (for whatever reason - like a debugger can inspect a
> core dump), and moreover to convert between old and new formats when moving
> from older to newer kernels.
>
> By doing so, we avoid many API issues - design, complexity, contents, and
> the amount of interfaces to be added.

Userspace is expected to inspect or convert the binary data. How does
that truly avoid many of the API issues mentioned above? If it's really
supposed to be a minimal API then the binary should be considered opaque
and userspace tools which inspect or convert these binaries should be
considered unreliable hacks at best. Otherwise it seems to me that it
has most of the familiar problems associated with a kernel/userspace API
-- including an obligation to support it.

Cheers,
-Matt Helsley

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

---

Subject: Re: [RFC PATCH 0/5] Resend -v2 - Use procfs to change a syscall behavior
Posted by ebiederm on Fri, 18 Jul 2008 02:40:45 GMT
View Forum Message <> Reply to Message

Oren Laadan <orenl@cs.columbia.edu> writes:

> I think the kernel space vs. user space must be the first issue on our
> table to solve, as it has a wide impact on the rest of the work.

We first need to talk about what kinds of problems we are trying to solve. If we don't agree what the problem is I expect we will have a hard time agreeing on a solution.

For example we are using namespaces now instead of the potentially simpler isolation mechanism of Vserver because checkpoint/restart could not be done with the Vserver approach.

The use case that I expect we all have in common is migrating an isolated container from one machine to another transparent to applications. Except those that directly access the hardware at which point we can treat it as a hotplug event from the perspective of userspace.

There are several other interesting use cases that I think we should solve if possible.
- Live/Incremental migration.
- Remote fork. Which can be seen as an extreme case of migrating only a partial container.
- A checkpoint that can be restarted multiple times and work properly. Which means you need to include the state of the filesystem.
- A distributed checkpoint of multiple containers at the same time.

Given how brutally hard and inefficient it is to restore a checkpoint using the existing system calls even with namespaces in the kernel. We can pretty much rule that implementation out as it does not match our efficiency criteria, and likely isn't especially maintainble either.

On the maintenance side we can generally rule out an out of tree module.
As that does not afford visible to people changing a subsystem that the
checkpoint/restart code needs to change as well.

I believe the live migration will have the most stringent performance
requirements and at the same time be one of the most useful features,
as it immediately improve maintenance of clusters.

In the extreme case of a distributed checkpoint the kernel simply does
not have enough state so we need user space code coordinating all of
the pieces.

For a multi-start checkpoint I expect userspace will be coordinating
filesystem snapshots and checkpoints.

Eric

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

---

Subject: Re: [RFC PATCH 0/5] Resend -v2 - Use procfs to change a syscall behavior
Posted by ebiederm on Fri, 18 Jul 2008 02:49:05 GMT
View Forum Message <> Reply to Message

Matt Helsley <matthltc@us.ibm.com> writes:

> On Thu, 2008-07-17 at 18:42 -0400, Oren Laadan wrote:
>>
>> My question is why build a set of interfaces to export this and that from
>> the kernel to user space ? if a kernel implementation (with minimal user
>> space support) is chosen, then information extraction (and restoration) is
>> straightforward and we don't get ourselves tied until the end of times to
>> API exported to userland.
>
> That still seems like an API exported to userland. It just combines the
> data into one block rather than distributing it amongst a bunch of
> pseudo-filesystems. Does this form of API really free us from always
> supporting it in the future?

A larger granularity reduces the support burden. You don't wind up
introducing a bunch of little system calls that you only use for
restore. You introduce one that does exactly what you need it to do.
Because you know it is only used in checkpoint/restart conditions you
can make assumptions about the users and have more freedom.

Yes it would still be a user/kernel interface.

If we abstract it something like binformats are abstracted we
may eventually be able to stop including an old format that no
one uses anymore.

>
> Userspace is expected to inspect or convert the binary data. How does
> that truly avoid many of the API issues mentioned above? If it's really
> supposed to be a minimal API then the binary should be considered opaque
> and userspace tools which inspect or convert these binaries should be
> considered unreliable hacks at best. Otherwise it seems to me that it
> has most of the familiar problems associated with a kernel/userspace API
> -- including an obligation to support it.

The best precedent we have for something like this today is the core
dump. That is a single process and does not do well at tying multiple
processes together. Even though you can inspect a core dump there is
still a lot of freedom in the implementation that we would not have
in a more general API.

As for userspace converting old data to new data. I'm not sold on the
idea yet. It is a good tool to plan on, but I'm not yet convinced
that it is necessary, at least when moving from older to newer kernels.
I expect newer kernels to have state that the older kernels don't know
how to handle, so we would at least need to strip that out.

Eric