Subject: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by Nadia Derbey on Thu, 03 Jul 2008 14:40:13 GMT
View Forum Message <> Reply to Message

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-rc5-mm3 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 2.6.25-rc3-mm2, 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

--

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

---

Subject: [RFC PATCH 1/5] adds the procfs facilities
Posted by Nadia Derbey on Thu, 03 Jul 2008 14:40:14 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 |   35 ++++++++
 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, 281 insertions(+), 1 deletion(-)
```

Index: linux-2.6.26-rc5-mm3/include/linux/sched.h
===
--- linux-2.6.26-rc5-mm3.orig/include/linux/sched.h 2008-06-25 17:10:38.000000000 +0200

```
+++ linux-2.6.26-rc5-mm3/include/linux/sched.h 2008-06-27 14:18:56.000000000 +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>

@@ -1312,6 +1313,11 @@ 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-rc5-mm3/include/linux/next_syscall_data.h
===================================================================
--- /dev/null 1970-01-01 00:00:00.000000000 +0000
+++ linux-2.6.26-rc5-mm3/include/linux/next_syscall_data.h 2008-07-01 10:25:48.000000000
+0200
@@ -0,0 +1,35 @@
+/*
+ * include/linux/next_syscall_data.h
+ *
+ * Definitions to support fixed data for next syscall to be called. The
+ * following is supported today:
+ *    . object creation with a predefined id.
+ *
+ */
+
+#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 int 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-rc5-mm3/fs/proc/base.c
===================================================================
--- linux-2.6.26-rc5-mm3.orig/fs/proc/base.c 2008-06-25 17:11:04.000000000 +0200
+++ linux-2.6.26-rc5-mm3/fs/proc/base.c 2008-07-01 09:09:30.000000000 +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-rc5-mm3/kernel/Makefile
===================================================================
--- linux-2.6.26-rc5-mm3.orig/kernel/Makefile 2008-06-25 17:10:41.000000000 +0200
+++ linux-2.6.26-rc5-mm3/kernel/Makefile 2008-06-27 09:03:01.000000000 +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-rc5-mm3/kernel/next_syscall_data.c
===================================================================
--- /dev/null 1970-01-01 00:00:00.000000000 +0000
+++ linux-2.6.26-rc5-mm3/kernel/next_syscall_data.c 2008-07-01 10:39:43.000000000 +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);
+ 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);
+}
+
+int reset_next_syscall_data(struct task_struct *task)
+{
+ struct next_syscall_data *nsd;
+
+ nsd = task->nsd;
```

```
+ if (!nsd)
+  return 0;
+
+ task->nsd = NULL;
+ kfree(nsd);
+ return 0;
+}
+
+#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)))
+  return reset_next_syscall_data(task);
+
+ return -EINVAL;
+}
Index: linux-2.6.26-rc5-mm3/kernel/fork.c
===================================================================
--- linux-2.6.26-rc5-mm3.orig/kernel/fork.c 2008-06-25 17:10:41.000000000 +0200
+++ linux-2.6.26-rc5-mm3/kernel/fork.c 2008-07-01 10:25:46.000000000 +0200
@@ -1077,6 +1077,8 @@ 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-rc5-mm3/fs/exec.c
===================================================================
--- linux-2.6.26-rc5-mm3.orig/fs/exec.c 2008-06-25 17:11:05.000000000 +0200
+++ linux-2.6.26-rc5-mm3/fs/exec.c 2008-06-27 14:53:08.000000000 +0200
@@ -1014,6 +1014,12 @@ 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-rc5-mm3/kernel/exit.c
===================================================================
--- linux-2.6.26-rc5-mm3.orig/kernel/exit.c 2008-06-25 17:10:41.000000000 +0200
+++ linux-2.6.26-rc5-mm3/kernel/exit.c 2008-06-27 14:57:55.000000000 +0200
@@ -1069,6 +1069,10 @@ 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 Thu, 03 Jul 2008 14:40:15 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 | 38 ++++++++++++++++++++++++++++++++++++++--------
 2 files changed, 45 insertions(+), 10 deletions(-)

Index: linux-2.6.26-rc5-mm3/include/linux/next_syscall_data.h
===
--- linux-2.6.26-rc5-mm3.orig/include/linux/next_syscall_data.h 2008-07-01 10:25:48.000000000 +0200
+++ linux-2.6.26-rc5-mm3/include/linux/next_syscall_data.h 2008-07-01 11:35:11.000000000 +0200
@@ -3,7 +3,8 @@
  *
  * Definitions to support fixed data for next syscall to be called. The
  * following is supported today:
- *   . object creation with a predefined id.
+ *   . object creation with a predefined id
+ *   . for a sysv ipc object
  *
  */

@@ -16,13 +17,25 @@
  * 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 int reset_next_syscall_data(struct task_struct *);
Index: linux-2.6.26-rc5-mm3/ipc/util.c
===================================================================
--- linux-2.6.26-rc5-mm3.orig/ipc/util.c 2008-07-01 10:25:48.000000000 +0200
+++ linux-2.6.26-rc5-mm3/ipc/util.c 2008-07-01 10:41:36.000000000 +0200
@@ -266,20 +266,42 @@ 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 (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;
+
+  if (next_id !=
+      (new_lid + (next_id / SEQ_MULTIPLIER) * 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 = next_id / SEQ_MULTIPLIER;
+  reset_next_syscall_data(current);
+ } 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 3/5] use next syscall data to predefine process ids
Posted by Nadia Derbey on Thu, 03 Jul 2008 14:40:16 GMT
View Forum Message <> Reply to Message

[PATCH 03/05]

This patch uses the value written into the next_syscall_data proc file
as a target upid nr for the next process to be created.
The following syscalls have a new behavior if next_syscall_data is set:
. fork()
. vfork()
. clone()

In the current version, if the process belongs to nested namespaces, only
the upper namespace level upid nr is allowed to be predefined, since there
is not yet a way to take a snapshot of upid nrs at all namespaces levels.

But this can easily be extended in the future.

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

```
 include/linux/next_syscall_data.h |    2
 include/linux/pid.h               |    2
```

```
 kernel/fork.c                 |    3 -
 kernel/pid.c                  |  111 +++++++++++++++++++++++++++++++------
 4 files changed, 98 insertions(+), 20 deletions(-)

Index: linux-2.6.26-rc5-mm3/kernel/pid.c
===================================================================
--- linux-2.6.26-rc5-mm3.orig/kernel/pid.c 2008-07-01 10:25:46.000000000 +0200
+++ linux-2.6.26-rc5-mm3/kernel/pid.c 2008-07-01 11:25:38.000000000 +0200
@@ -122,6 +122,26 @@ static void free_pidmap(struct upid *upi
  atomic_inc(&map->nr_free);
 }

+static inline int alloc_pidmap_page(struct pidmap *map)
+{
+ if (unlikely(!map->page)) {
+  void *page = kzalloc(PAGE_SIZE, GFP_KERNEL);
+  /*
+   * Free the page if someone raced with us
+   * installing it:
+   */
+  spin_lock_irq(&pidmap_lock);
+  if (map->page)
+   kfree(page);
+  else
+   map->page = page;
+  spin_unlock_irq(&pidmap_lock);
+  if (unlikely(!map->page))
+   return -1;
+ }
+ return 0;
+}
+
 static int alloc_pidmap(struct pid_namespace *pid_ns)
 {
  int i, offset, max_scan, pid, last = pid_ns->last_pid;
@@ -134,21 +154,8 @@ static int alloc_pidmap(struct pid_names
  map = &pid_ns->pidmap[pid/BITS_PER_PAGE];
  max_scan = (pid_max + BITS_PER_PAGE - 1)/BITS_PER_PAGE - !offset;
  for (i = 0; i <= max_scan; ++i) {
-  if (unlikely(!map->page)) {
-   void *page = kzalloc(PAGE_SIZE, GFP_KERNEL);
-   /*
-    * Free the page if someone raced with us
-    * installing it:
-    */
-   spin_lock_irq(&pidmap_lock);
-   if (map->page)
-    kfree(page);
```

```
-   else
-    map->page = page;
-   spin_unlock_irq(&pidmap_lock);
-   if (unlikely(!map->page))
-    break;
-  }
+  if (unlikely(alloc_pidmap_page(map)))
+   break;
   if (likely(atomic_read(&map->nr_free))) {
    do {
     if (!test_and_set_bit(offset, map->page)) {
@@ -182,6 +189,33 @@ static int alloc_pidmap(struct pid_names
  return -1;
 }

+/*
+ * Return 0 if successful (i.e. next_nr could be assigned as a upid nr).
+ * -errno else
+ */
+static int alloc_fixed_pidmap(struct pid_namespace *pid_ns, int next_nr)
+{
+ int offset;
+ struct pidmap *map;
+
+ if (next_nr < RESERVED_PIDS || next_nr >= pid_max)
+  return -EINVAL;
+
+ map = &pid_ns->pidmap[next_nr / BITS_PER_PAGE];
+
+ if (unlikely(alloc_pidmap_page(map)))
+  return -ENOMEM;
+
+ offset = next_nr & BITS_PER_PAGE_MASK;
+ if (test_and_set_bit(offset, map->page))
+  return -EBUSY;
+
+ atomic_dec(&map->nr_free);
+ pid_ns->last_pid = max(pid_ns->last_pid, next_nr);
+
+ return 0;
+}
+
 int next_pidmap(struct pid_namespace *pid_ns, int last)
 {
  int offset;
@@ -239,7 +273,25 @@ void free_pid(struct pid *pid)
  call_rcu(&pid->rcu, delayed_put_pid);
 }
```

```
-struct pid *alloc_pid(struct pid_namespace *ns)
+/*
+ * Sets a predefined upid nr for the process' upper namespace level
+ */
+static int set_predefined_pid(struct pid_namespace *ns, struct pid *pid,
+    int next_nr)
+{
+ int i = ns->level;
+ int rc;
+
+ rc = alloc_fixed_pidmap(ns, next_nr);
+ if (rc < 0)
+  return rc;
+
+ pid->numbers[i].nr = next_nr;
+ pid->numbers[i].ns = ns;
+ return 0;
+}
+
+struct pid *alloc_pid(struct pid_namespace *ns, int *retval)
 {
  struct pid *pid;
  enum pid_type type;
@@ -247,12 +299,37 @@ struct pid *alloc_pid(struct pid_namespa
  struct pid_namespace *tmp;
  struct upid *upid;

+ *retval = -ENOMEM;
  pid = kmem_cache_alloc(ns->pid_cachep, GFP_KERNEL);
  if (!pid)
   goto out;

  tmp = ns;
- for (i = ns->level; i >= 0; i--) {
+ i = ns->level;
+ if (next_data_set(current)) {
+  /*
+   * There is a upid nr specified, use it instead of letting
+   * the kernel chose it for us.
+   */
+  int next_nr = get_next_data(current);
+  int rc;
+
+  rc = set_predefined_pid(tmp, pid, next_nr);
+  if (rc < 0) {
+   *retval = rc;
+   goto out_free;
```

```
+  }
+  /* Go up one level */
+  tmp = tmp->parent;
+  i--;
+  reset_next_syscall_data(current);
+ }
+
+ /*
+  * Let the lower levels upid nrs be automatically allocated
+  */
+ *retval = -ENOMEM;
+ for ( ; i >= 0; i--) {
   nr = alloc_pidmap(tmp);
   if (nr < 0)
    goto out_free;
Index: linux-2.6.26-rc5-mm3/include/linux/pid.h
===================================================================
--- linux-2.6.26-rc5-mm3.orig/include/linux/pid.h 2008-07-01 10:25:46.000000000 +0200
+++ linux-2.6.26-rc5-mm3/include/linux/pid.h 2008-07-01 10:49:07.000000000 +0200
@@ -121,7 +121,7 @@ extern struct pid *find_get_pid(int nr);
 extern struct pid *find_ge_pid(int nr, struct pid_namespace *);
 int next_pidmap(struct pid_namespace *pid_ns, int last);

-extern struct pid *alloc_pid(struct pid_namespace *ns);
+extern struct pid *alloc_pid(struct pid_namespace *, int *);
 extern void free_pid(struct pid *pid);

 /*
Index: linux-2.6.26-rc5-mm3/kernel/fork.c
===================================================================
--- linux-2.6.26-rc5-mm3.orig/kernel/fork.c 2008-07-01 10:25:46.000000000 +0200
+++ linux-2.6.26-rc5-mm3/kernel/fork.c 2008-07-01 10:49:07.000000000 +0200
@@ -1110,8 +1110,7 @@ static struct task_struct *copy_process(
   goto bad_fork_cleanup_io;

  if (pid != &init_struct_pid) {
-  retval = -ENOMEM;
-  pid = alloc_pid(task_active_pid_ns(p));
+  pid = alloc_pid(task_active_pid_ns(p), &retval);
   if (!pid)
    goto bad_fork_cleanup_io;

Index: linux-2.6.26-rc5-mm3/include/linux/next_syscall_data.h
===================================================================
--- linux-2.6.26-rc5-mm3.orig/include/linux/next_syscall_data.h 2008-07-01 10:41:36.000000000
+0200
+++ linux-2.6.26-rc5-mm3/include/linux/next_syscall_data.h 2008-07-01 11:09:35.000000000
+0200
```

```
@@ -5,6 +5,7 @@
  * following is supported today:
  *    . object creation with a predefined id
  *          . for a sysv ipc object
+ *          . for a process
  *
  */

@@ -19,6 +20,7 @@
  * For example, it can be used to pre-set the id of the object to be created
  * by next syscall. The following syscalls support this feature:
  *    . msgget(), semget(), shmget()
+ *    . fork(), vfork(), clone()
  */
 struct next_syscall_data {
  int ndata;

--
```

_____

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

---

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

Posted by Nadia Derbey on Thu, 03 Jul 2008 14:40:17 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, 62 insertions(+), 4 deletions(-)

Index: linux-2.6.26-rc5-mm3/include/linux/next_syscall_data.h
===================================================================
--- linux-2.6.26-rc5-mm3.orig/include/linux/next_syscall_data.h 2008-07-01 12:07:46.000000000
+0200
+++ linux-2.6.26-rc5-mm3/include/linux/next_syscall_data.h 2008-07-01 12:07:50.000000000
+0200
@@ -6,7 +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
@@ -21,6 +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;
@@ -36,6 +40,12 @@ 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);
Index: linux-2.6.26-rc5-mm3/ipc/msg.c
===================================================================
```

```
--- linux-2.6.26-rc5-mm3.orig/ipc/msg.c 2008-07-01 11:09:35.000000000 +0200
+++ linux-2.6.26-rc5-mm3/ipc/msg.c 2008-07-01 12:07:50.000000000 +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 (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-rc5-mm3/ipc/sem.c
===================================================================
--- linux-2.6.26-rc5-mm3.orig/ipc/sem.c 2008-07-01 11:09:35.000000000 +0200
+++ linux-2.6.26-rc5-mm3/ipc/sem.c 2008-07-01 12:07:50.000000000 +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 (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-rc5-mm3/ipc/shm.c
===================================================================
--- linux-2.6.26-rc5-mm3.orig/ipc/shm.c 2008-07-01 11:09:35.000000000 +0200
+++ linux-2.6.26-rc5-mm3/ipc/shm.c 2008-07-01 12:07:50.000000000 +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 (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: [RFC PATCH 5/5] use next syscall data to predefine the file descriptor value

[PATCH 05/05]

This patch uses the value written into the next_syscall_data proc file
as a target file descriptor for the next file to be opened.

This makes it easy to restart a process with the same fds as the ones it was
using during the checkpoint phase, instead of 1. opening the file, 2. dup2'ing
the open file descriptor.

The following syscalls are impacted if next_syscall_data is set:
. open()
. openat()

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

---
 fs/open.c | 58 ++-
 1 file changed, 57 insertions(+), 1 deletion(-)

Index: linux-2.6.26-rc5-mm3/fs/open.c
===
--- linux-2.6.26-rc5-mm3.orig/fs/open.c 2008-06-25 17:11:06.000000000 +0200
+++ linux-2.6.26-rc5-mm3/fs/open.c 2008-07-01 17:51:53.000000000 +0200
@@ -967,6 +967,55 @@ struct file *dentry_open(struct dentry *
 EXPORT_SYMBOL(dentry_open);

 /*
+ * Marks a given file descriptor entry as busy (should not be busy when this
+ * routine is called.
+ *
+ * files->next_fd is not updated: this lets the potentially created hole be
+ * filled up on next calls to get_unused_fd_flags.
+ *
+ * Returns the specified fd if successful, -errno else.
+ */
+static int get_predefined_fd_flags(int fd, int flags)
+{
+ struct files_struct *files = current->files;
+ int error;
+ struct fdtable *fdt;
+
+ error = -EINVAL;
+ if (fd < 0)
+ goto out;
+
+ error = -EMFILE;

```
+ if (fd >= current->signal->rlim[RLIMIT_NOFILE].rlim_cur)
+  goto out;
+
+ spin_lock(&files->file_lock);
+ fdt = files_fdtable(files);
+
+ error = expand_files(files, fd);
+ if (error < 0)
+  goto out_unlock;
+
+ error = -EBUSY;
+ if (FD_ISSET(fd, fdt->open_fds))
+  goto out_unlock;
+
+ FD_SET(fd, fdt->open_fds);
+ if (flags & O_CLOEXEC)
+  FD_SET(fd, fdt->close_on_exec);
+ else
+  FD_CLR(fd, fdt->close_on_exec);
+
+ BUG_ON(fdt->fd[fd] != NULL);
+
+ error = fd;
+out_unlock:
+ spin_unlock(&files->file_lock);
+out:
+ return error;
+}
+
+/*
  * Find an empty file descriptor entry, and mark it busy.
  */
 int get_unused_fd_flags(int flags)
@@ -1081,7 +1130,14 @@ long do_sys_open(int dfd, const char __u
  int fd = PTR_ERR(tmp);

  if (!IS_ERR(tmp)) {
-  fd = get_unused_fd_flags(flags);
+  if (next_data_set(current)) {
+   int next_fd = get_next_data(current);
+
+   fd = get_predefined_fd_flags(next_fd, flags);
+   reset_next_syscall_data(current);
+  } else
+   fd = get_unused_fd_flags(flags);
+
   if (fd >= 0) {
    struct file *f = do_filp_open(dfd, tmp, flags, mode);
```

if (IS_ERR(f)) {

--

_____

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

---

Subject: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by Pavel Machek on Fri, 04 Jul 2008 10:27:02 GMT
View Forum Message <> Reply to Message

Hi!

> 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.

Yes, and I believe that's better than...

> 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().

...bloat task struct and

> 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.

...introducing this kind of uglyness.

Actually, there were proposals for sys_indirect(), which is slightly
less ugly, but IIRC we ended up with adding syscalls, too.
Pavel
--
(english) http://www.livejournal.com/~pavelmachek
(cesky, pictures) http://atrey.karlin.mff.cuni.cz/~pavel/picture/horses/blog.html

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

---

Subject: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by Nadia Derbey on Fri, 04 Jul 2008 12:07:18 GMT
View Forum Message <> Reply to Message

Pavel Machek wrote:
> Hi!
>
>
>>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.
>
>
> Yes, and I believe that's better than...
>
>
>>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().
>
>
> ...bloat task struct and
>
>
>>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.
>
>
> ...introducing this kind of uglyness.
>
> Actually, there were proposals for sys_indirect(), which is slightly
> less ugly, but IIRC we ended up with adding syscalls, too.
> Pavel

Pavel,

I had a look at the lwn.net article that describes the sys_indirect()
interface.
It does exactly what we need here, so I do like it, but it has the same
drawbacks as the one you're complaining about:
. a new field is needed in the task structure
. looks like many people found it ugly...

Now, coming back to what I'm proposing: what we need is actually to
change the behavior of *existing* syscalls, since we are in a very
particular context (restarting an application).

Defining brand new syscalls is very touchy: needs to be careful about
the interface + I can't imagine the number of syscalls that would be needed.

Now, since we do have a set of available syscalls, I think it's much
easier to change their behavior depending on a field being set in the

task structure.

I agree with you that the interface is not that nice, so what about
proposing a single syscall that would set the next_syscall_data field in
the task structure (instead of setting it through procfs).
It's true that this makes us end up with a "2 passes" sys_indirect()
(i.e. 2 syscalls called instead of a single one), but it is much
simpler. And may be the induced performance overhead would not be that
important since we are, again, in a particular context (restarting an
application)?

Regards,
Nadia

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

---

Subject: Re: [RFC PATCH 1/5] adds the procfs facilities
Posted by serue on Mon, 07 Jul 2008 18:30:30 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>
>
> ---
> fs/exec.c | 6 +
> fs/proc/base.c | 75 ++++++++++++++++++++

> include/linux/next_syscall_data.h | 35 ++++++++
> 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, 281 insertions(+), 1 deletion(-)
>
> Index: linux-2.6.26-rc5-mm3/include/linux/sched.h
> ===
> --- linux-2.6.26-rc5-mm3.orig/include/linux/sched.h 2008-06-25 17:10:38.000000000 +0200
> +++ linux-2.6.26-rc5-mm3/include/linux/sched.h 2008-06-27 14:18:56.000000000 +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>
>
> @@ -1312,6 +1313,11 @@ 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-rc5-mm3/include/linux/next_syscall_data.h
> ===
> --- /dev/null 1970-01-01 00:00:00.000000000 +0000
> +++ linux-2.6.26-rc5-mm3/include/linux/next_syscall_data.h 2008-07-01 10:25:48.000000000
+0200
> @@ -0,0 +1,35 @@
> +/*
> + * include/linux/next_syscall_data.h
> + *
> + * Definitions to support fixed data for next syscall to be called. The
> + * following is supported today:
> + *   . object creation with a predefined id.
> + *
> + */
> +
> +#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 int 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-rc5-mm3/fs/proc/base.c
> ===================================================================
> --- linux-2.6.26-rc5-mm3.orig/fs/proc/base.c 2008-06-25 17:11:04.000000000 +0200
> +++ linux-2.6.26-rc5-mm3/fs/proc/base.c 2008-07-01 09:09:30.000000000 +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-rc5-mm3/kernel/Makefile
> ===
> --- linux-2.6.26-rc5-mm3.orig/kernel/Makefile 2008-06-25 17:10:41.000000000 +0200
> +++ linux-2.6.26-rc5-mm3/kernel/Makefile 2008-06-27 09:03:01.000000000 +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-rc5-mm3/kernel/next_syscall_data.c
> ===
> --- /dev/null 1970-01-01 00:00:00.000000000 +0000
> +++ linux-2.6.26-rc5-mm3/kernel/next_syscall_data.c 2008-07-01 10:39:43.000000000 +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);
```

Shouldn't you also reset task->nsd to NULL here? :-)

```
> + 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);
> +}
> +
> +int reset_next_syscall_data(struct task_struct *task)

Why have this return an int? It always returns 0, and callers ignore
the return value.

> +{
> + struct next_syscall_data *nsd;
> +
> + nsd = task->nsd;
> + if (!nsd)
> + return 0;
> +
> + task->nsd = NULL;
> + kfree(nsd);
> + return 0;
> +}
> +
> +#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)))
> + return reset_next_syscall_data(task);
> +
> + return -EINVAL;
> +}
> Index: linux-2.6.26-rc5-mm3/kernel/fork.c
> ===
> --- linux-2.6.26-rc5-mm3.orig/kernel/fork.c 2008-06-25 17:10:41.000000000 +0200
> +++ linux-2.6.26-rc5-mm3/kernel/fork.c 2008-07-01 10:25:46.000000000 +0200
> @@ -1077,6 +1077,8 @@ 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-rc5-mm3/fs/exec.c
> ===
> --- linux-2.6.26-rc5-mm3.orig/fs/exec.c 2008-06-25 17:11:05.000000000 +0200
> +++ linux-2.6.26-rc5-mm3/fs/exec.c 2008-06-27 14:53:08.000000000 +0200
> @@ -1014,6 +1014,12 @@ 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-rc5-mm3/kernel/exit.c
> ===
> --- linux-2.6.26-rc5-mm3.orig/kernel/exit.c 2008-06-25 17:10:41.000000000 +0200
> +++ linux-2.6.26-rc5-mm3/kernel/exit.c 2008-06-27 14:57:55.000000000 +0200
> @@ -1069,6 +1069,10 @@ 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 Mon, 07 Jul 2008 18:35:12 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>
>
> ---
> include/linux/next_syscall_data.h | 17 ++++++++++++++--
> ipc/util.c | 38 ++++++++++++++++++++++++++++++++++++++--------
> 2 files changed, 45 insertions(+), 10 deletions(-)
>
> Index: linux-2.6.26-rc5-mm3/include/linux/next_syscall_data.h
> ===
> --- linux-2.6.26-rc5-mm3.orig/include/linux/next_syscall_data.h 2008-07-01 10:25:48.000000000
+0200
> +++ linux-2.6.26-rc5-mm3/include/linux/next_syscall_data.h 2008-07-01 11:35:11.000000000
+0200
> @@ -3,7 +3,8 @@
>   *
>   * Definitions to support fixed data for next syscall to be called. The
>   * following is supported today:
> - *   . object creation with a predefined id.
> + *   . object creation with a predefined id
> + *   . for a sysv ipc object
>   *
>   */

>
> @@ -16,13 +17,25 @@
>   * 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 int reset_next_syscall_data(struct task_struct *);
> Index: linux-2.6.26-rc5-mm3/ipc/util.c
> ===
> --- linux-2.6.26-rc5-mm3.orig/ipc/util.c 2008-07-01 10:25:48.000000000 +0200
> +++ linux-2.6.26-rc5-mm3/ipc/util.c 2008-07-01 10:41:36.000000000 +0200
> @@ -266,20 +266,42 @@ 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 (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;
> +
> + if (next_id !=
> +   (new_lid + (next_id / SEQ_MULTIPLIER) * SEQ_MULTIPLIER))
> +   return -EINVAL;

You're leaving the next_data info set on error. Should we clear it?

I think it seems more reasonable to clear it on error and just expect
the application, if it wants to retry, re-set it to the desired id
before retry.

```
> +
> +  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 = next_id / SEQ_MULTIPLIER;
> +  reset_next_syscall_data(current);
> + } 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 3/5] use next syscall data to predefine process ids
Posted by serue on Mon, 07 Jul 2008 18:54:24 GMT
View Forum Message <> Reply to Message

Quoting Nadia.Derbey@bull.net (Nadia.Derbey@bull.net):
> [PATCH 03/05]
>
> This patch uses the value written into the next_syscall_data proc file
> as a target upid nr for the next process to be created.
> The following syscalls have a new behavior if next_syscall_data is set:
> . fork()
> . vfork()
> . clone()
>
> In the current version, if the process belongs to nested namespaces, only
> the upper namespace level upid nr is allowed to be predefined, since there
> is not yet a way to take a snapshot of upid nrs at all namespaces levels.
>
> But this can easily be extended in the future.

Good point, we will want to discuss the right way to dump that data. Do
we add a new file under /proc/<pid>, use /proc/pid/status, or find some
other way?

> Signed-off-by: Nadia Derbey <Nadia.Derbey@bull.net>
>
> ---
> include/linux/next_syscall_data.h | 2
> include/linux/pid.h | 2
> kernel/fork.c | 3 -
> kernel/pid.c | 111 ++++++++++++++++++++++++++++++++++++++------
> 4 files changed, 98 insertions(+), 20 deletions(-)
>
> Index: linux-2.6.26-rc5-mm3/kernel/pid.c
> ===
> --- linux-2.6.26-rc5-mm3.orig/kernel/pid.c 2008-07-01 10:25:46.000000000 +0200
> +++ linux-2.6.26-rc5-mm3/kernel/pid.c 2008-07-01 11:25:38.000000000 +0200
> @@ -122,6 +122,26 @@ static void free_pidmap(struct upid *upi
> atomic_inc(&map->nr_free);
> }
>
> +static inline int alloc_pidmap_page(struct pidmap *map)
> +{
> + if (unlikely(!map->page)) {
> + void *page = kzalloc(PAGE_SIZE, GFP_KERNEL);
> + /*
> + * Free the page if someone raced with us
> + * installing it:
> + */

```
> +  spin_lock_irq(&pidmap_lock);
> +  if (map->page)
> +   kfree(page);
> +  else
> +   map->page = page;
> +  spin_unlock_irq(&pidmap_lock);
> +  if (unlikely(!map->page))
> +   return -1;
> + }
> + return 0;
> +}
> +
>  static int alloc_pidmap(struct pid_namespace *pid_ns)
>  {
>   int i, offset, max_scan, pid, last = pid_ns->last_pid;
> @@ -134,21 +154,8 @@ static int alloc_pidmap(struct pid_names
>   map = &pid_ns->pidmap[pid/BITS_PER_PAGE];
>   max_scan = (pid_max + BITS_PER_PAGE - 1)/BITS_PER_PAGE - !offset;
>   for (i = 0; i <= max_scan; ++i) {
> -  if (unlikely(!map->page)) {
> -   void *page = kzalloc(PAGE_SIZE, GFP_KERNEL);
> -   /*
> -    * Free the page if someone raced with us
> -    * installing it:
> -    */
> -   spin_lock_irq(&pidmap_lock);
> -   if (map->page)
> -    kfree(page);
> -   else
> -    map->page = page;
> -   spin_unlock_irq(&pidmap_lock);
> -   if (unlikely(!map->page))
> -    break;
> -  }
> +  if (unlikely(alloc_pidmap_page(map)))
> +   break;
>    if (likely(atomic_read(&map->nr_free))) {
>     do {
>      if (!test_and_set_bit(offset, map->page)) {
> @@ -182,6 +189,33 @@ static int alloc_pidmap(struct pid_names
>   return -1;
>  }
>
> +/*
> + * Return 0 if successful (i.e. next_nr could be assigned as a upid nr).
> + * -errno else
> + */
> +static int alloc_fixed_pidmap(struct pid_namespace *pid_ns, int next_nr)
```

```
> +{
> + int offset;
> + struct pidmap *map;
> +
> + if (next_nr < RESERVED_PIDS || next_nr >= pid_max)
> +  return -EINVAL;
> +
> + map = &pid_ns->pidmap[next_nr / BITS_PER_PAGE];
> +
> + if (unlikely(alloc_pidmap_page(map)))
> +  return -ENOMEM;
> +
> + offset = next_nr & BITS_PER_PAGE_MASK;
> + if (test_and_set_bit(offset, map->page))
> +  return -EBUSY;
> +
> + atomic_dec(&map->nr_free);
> + pid_ns->last_pid = max(pid_ns->last_pid, next_nr);
> +
> + return 0;
> +}
> +
>  int next_pidmap(struct pid_namespace *pid_ns, int last)
>  {
>   int offset;
> @@ -239,7 +273,25 @@ void free_pid(struct pid *pid)
>   call_rcu(&pid->rcu, delayed_put_pid);
>  }
>
> -struct pid *alloc_pid(struct pid_namespace *ns)
> +/*
> + * Sets a predefined upid nr for the process' upper namespace level
> + */
> +static int set_predefined_pid(struct pid_namespace *ns, struct pid *pid,
> +    int next_nr)
> +{
> + int i = ns->level;
> + int rc;
> +
> + rc = alloc_fixed_pidmap(ns, next_nr);
> + if (rc < 0)
> +  return rc;
> +
> + pid->numbers[i].nr = next_nr;
> + pid->numbers[i].ns = ns;
> + return 0;
> +}
> +
```

> +struct pid *alloc_pid(struct pid_namespace *ns, int *retval)

Is there a reason why you can't return retval using
 return ERR_PTR(retval);
instead of using an additional argument? Then at copy_process,
after the call, do

 if (IS_ERR(pid))
  retval = PTR_ERR(pid);

?

> {
>   struct pid *pid;
>   enum pid_type type;
> @@ -247,12 +299,37 @@ struct pid *alloc_pid(struct pid_namespa
>   struct pid_namespace *tmp;
>   struct upid *upid;
>
> + *retval = -ENOMEM;
>   pid = kmem_cache_alloc(ns->pid_cachep, GFP_KERNEL);
>   if (!pid)
>   goto out;
>
>   tmp = ns;
> - for (i = ns->level; i >= 0; i--) {
> + i = ns->level;
> + if (next_data_set(current)) {
> + /*
> +   * There is a upid nr specified, use it instead of letting
> +   * the kernel chose it for us.
> +   */
> + int next_nr = get_next_data(current);
> + int rc;
> +
> + rc = set_predefined_pid(tmp, pid, next_nr);
> + if (rc < 0) {
> +   *retval = rc;
> +   goto out_free;

Again, I'd argue for resetting the syscall data on failure.

> + }
> + /* Go up one level */
> + tmp = tmp->parent;
> + i--;
> + reset_next_syscall_data(current);
> + }

```
> +
> + /*
> +  * Let the lower levels upid nrs be automatically allocated
> +  */
> + *retval = -ENOMEM;
> + for ( ; i >= 0; i--) {
>    nr = alloc_pidmap(tmp);
>    if (nr < 0)
>     goto out_free;
> Index: linux-2.6.26-rc5-mm3/include/linux/pid.h
> ===================================================================
> --- linux-2.6.26-rc5-mm3.orig/include/linux/pid.h 2008-07-01 10:25:46.000000000 +0200
> +++ linux-2.6.26-rc5-mm3/include/linux/pid.h 2008-07-01 10:49:07.000000000 +0200
> @@ -121,7 +121,7 @@ extern struct pid *find_get_pid(int nr);
>  extern struct pid *find_ge_pid(int nr, struct pid_namespace *);
>  int next_pidmap(struct pid_namespace *pid_ns, int last);
>
> -extern struct pid *alloc_pid(struct pid_namespace *ns);
> +extern struct pid *alloc_pid(struct pid_namespace *, int *);
>  extern void free_pid(struct pid *pid);
>
>  /*
> Index: linux-2.6.26-rc5-mm3/kernel/fork.c
> ===================================================================
> --- linux-2.6.26-rc5-mm3.orig/kernel/fork.c 2008-07-01 10:25:46.000000000 +0200
> +++ linux-2.6.26-rc5-mm3/kernel/fork.c 2008-07-01 10:49:07.000000000 +0200
> @@ -1110,8 +1110,7 @@ static struct task_struct *copy_process(
>    goto bad_fork_cleanup_io;
>
>   if (pid != &init_struct_pid) {
> -  retval = -ENOMEM;
> -  pid = alloc_pid(task_active_pid_ns(p));
> +  pid = alloc_pid(task_active_pid_ns(p), &retval);
>    if (!pid)
>     goto bad_fork_cleanup_io;
>
> Index: linux-2.6.26-rc5-mm3/include/linux/next_syscall_data.h
> ===================================================================
> --- linux-2.6.26-rc5-mm3.orig/include/linux/next_syscall_data.h 2008-07-01 10:41:36.000000000
+0200
> +++ linux-2.6.26-rc5-mm3/include/linux/next_syscall_data.h 2008-07-01 11:09:35.000000000
+0200
> @@ -5,6 +5,7 @@
>   * following is supported today:
>   *    . object creation with a predefined id
>   *         . for a sysv ipc object
> + *         . for a process
>   *
```

> */
>
> @@ -19,6 +20,7 @@
> * For example, it can be used to pre-set the id of the object to be created
> * by next syscall. The following syscalls support this feature:
> * . msgget(), semget(), shmget()
> + * . fork(), vfork(), clone()
> */
> struct next_syscall_data {
> int ndata;
>
> --

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

---

Subject: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by serue on Mon, 07 Jul 2008 19:01:19 GMT
View Forum Message <> Reply to Message

Quoting Pavel Machek (pavel@ucw.cz):
> Hi!
>
> > 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.

>
> Yes, and I believe that's better than...
>
> > 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().
>
> ...bloat task struct and
>
> > 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.
>
> ...introducing this kind of uglyness.
>
> Actually, there were proposals for sys_indirect(), which is slightly
> less ugly, but IIRC we ended up with adding syscalls, too.
> Pavel

Silly question...

Oren, would you object to defining sys_fork_with_id(),
sys_msgget_with_id(), and sys_semget_with_id()?

Eric, Pavel (Emelyanov), Dave, do you have preferences?

For the cases Nadia has implemented here I'd be tempted to side with
Pavel Machek, but once we get to things like open() and socket(), (a)
the # new syscalls starts to jump, and (b) the per-syscall api starts to
seem a lot more cumbersome.

-serge

Subject: Re: [RFC PATCH 1/5] adds the procfs facilities
Posted by Nadia Derbey on Tue, 08 Jul 2008 05:25:18 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> 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>
>>
>>---
>> fs/exec.c | 6 +
>> fs/proc/base.c | 75 ++++++++++++++++++
>> include/linux/next_syscall_data.h | 35 ++++++++
>> 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, 281 insertions(+), 1 deletion(-)
>>
>>Index: linux-2.6.26-rc5-mm3/include/linux/sched.h
>>===
>>--- linux-2.6.26-rc5-mm3.orig/include/linux/sched.h 2008-06-25 17:10:38.000000000 +0200
>>+++ linux-2.6.26-rc5-mm3/include/linux/sched.h 2008-06-27 14:18:56.000000000 +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>
>>
>>@@ -1312,6 +1313,11 @@ 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-rc5-mm3/include/linux/next_syscall_data.h
>>===
>>--- /dev/null 1970-01-01 00:00:00.000000000 +0000
>>+++ linux-2.6.26-rc5-mm3/include/linux/next_syscall_data.h 2008-07-01 10:25:48.000000000 +0200
>>@@ -0,0 +1,35 @@
>>+/*
>>+ * include/linux/next_syscall_data.h
>>+ *
>>+ * Definitions to support fixed data for next syscall to be called. The
>>+ * following is supported today:
>>+ *   . object creation with a predefined id.
>>+ *
>>+ */
>>+
>>+#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 int 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-rc5-mm3/fs/proc/base.c
>>===================================================================
>>--- linux-2.6.26-rc5-mm3.orig/fs/proc/base.c 2008-06-25 17:11:04.000000000 +0200
>>+++ linux-2.6.26-rc5-mm3/fs/proc/base.c 2008-07-01 09:09:30.000000000 +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-rc5-mm3/kernel/Makefile
>>===================================================================
>>--- linux-2.6.26-rc5-mm3.orig/kernel/Makefile 2008-06-25 17:10:41.000000000 +0200
```

```
>>+++ linux-2.6.26-rc5-mm3/kernel/Makefile 2008-06-27 09:03:01.000000000 +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-rc5-mm3/kernel/next_syscall_data.c
>>===================================================================
>>--- /dev/null 1970-01-01 00:00:00.000000000 +0000
>>+++ linux-2.6.26-rc5-mm3/kernel/next_syscall_data.c 2008-07-01 10:39:43.000000000 +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);
>

Serge,

Thanks for reviewing this so fast!

>
> Shouldn't you also reset task->nsd to NULL here? :-)

Oh yes!

>
>
>>+ 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);
>>+}
>>+
>>+int reset_next_syscall_data(struct task_struct *task)
>
>
> Why have this return an int? It always returns 0, and callers ignore
> the return value.
>

You're right, will change it to a void.

>
>>+{
>>+ struct next_syscall_data *nsd;
>>+
>>+ nsd = task->nsd;
>>+ if (!nsd)
>>+ return 0;
>>+
>>+ task->nsd = NULL;
>>+ kfree(nsd);
>>+ return 0;
>>+}
>>+
>>+#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)))
>>+ return reset_next_syscall_data(task);
>>+
>>+ return -EINVAL;
>>+}
>>Index: linux-2.6.26-rc5-mm3/kernel/fork.c
>>===
>>--- linux-2.6.26-rc5-mm3.orig/kernel/fork.c 2008-06-25 17:10:41.000000000 +0200

>>+++ linux-2.6.26-rc5-mm3/kernel/fork.c 2008-07-01 10:25:46.000000000 +0200
>>@@ -1077,6 +1077,8 @@ 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-rc5-mm3/fs/exec.c
>>===
>>--- linux-2.6.26-rc5-mm3.orig/fs/exec.c 2008-06-25 17:11:05.000000000 +0200
>>+++ linux-2.6.26-rc5-mm3/fs/exec.c 2008-06-27 14:53:08.000000000 +0200
>>@@ -1014,6 +1014,12 @@ 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-rc5-mm3/kernel/exit.c
>>===
>>--- linux-2.6.26-rc5-mm3.orig/kernel/exit.c 2008-06-25 17:10:41.000000000 +0200
>>+++ linux-2.6.26-rc5-mm3/kernel/exit.c 2008-06-27 14:57:55.000000000 +0200
>>@@ -1069,6 +1069,10 @@ 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 Nadia Derbey on Tue, 08 Jul 2008 05:30:37 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> 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>
>>
>>---
>> include/linux/next_syscall_data.h | 17 ++++++++++++++--
>> ipc/util.c | 38 ++++++++++++++++++++++++++++++++++++++--------
>> 2 files changed, 45 insertions(+), 10 deletions(-)
>>
>>Index: linux-2.6.26-rc5-mm3/include/linux/next_syscall_data.h
>>===
>>--- linux-2.6.26-rc5-mm3.orig/include/linux/next_syscall_data.h 2008-07-01
10:25:48.000000000 +0200
>>+++ linux-2.6.26-rc5-mm3/include/linux/next_syscall_data.h 2008-07-01 11:35:11.000000000
+0200
>>@@ -3,7 +3,8 @@
>> *
>> * Definitions to support fixed data for next syscall to be called. The
>> * following is supported today:
>>- * . object creation with a predefined id.
>>+ * . object creation with a predefined id
>>+ * . for a sysv ipc object
>> *
>> */
>>
>>@@ -16,13 +17,25 @@

```
>>  * 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 int reset_next_syscall_data(struct task_struct *);
>>Index: linux-2.6.26-rc5-mm3/ipc/util.c
>>===================================================================
>>--- linux-2.6.26-rc5-mm3.orig/ipc/util.c 2008-07-01 10:25:48.000000000 +0200
>>+++ linux-2.6.26-rc5-mm3/ipc/util.c 2008-07-01 10:41:36.000000000 +0200
>>@@ -266,20 +266,42 @@ 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 (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;
>>+
>>+  if (next_id !=
>>+      (new_lid + (next_id / SEQ_MULTIPLIER) * SEQ_MULTIPLIER))
>>+   return -EINVAL;
>
>
> You're leaving the next_data info set on error.  Should we clear it?
```

Well. I presently leave this cleaning up to the calling application. But
you're right, it is certainly cleaner to clear everything on error. The
only reason for this being that the uncleared data will be taken for
next syscall that is "next_syscall_data - sensitive".

>
> I think it seems more reasonable to clear it on error and just expect
> the application, if it wants to retry, re-set it to the desired id
> before retry.
>
>
>>+
>>+ 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 = next_id / SEQ_MULTIPLIER;
>>+ reset_next_syscall_data(current);
>>+ } 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();

>>
>>--
>
>
>

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

---

Subject: Re: [RFC PATCH 3/5] use next syscall data to predefine process ids
Posted by Nadia Derbey on Tue, 08 Jul 2008 05:44:22 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Nadia.Derbey@bull.net (Nadia.Derbey@bull.net):
>
>>[PATCH 03/05]
>>
>>This patch uses the value written into the next_syscall_data proc file
>>as a target upid nr for the next process to be created.
>>The following syscalls have a new behavior if next_syscall_data is set:
>>. fork()
>>. vfork()
>>. clone()
>>
>>In the current version, if the process belongs to nested namespaces, only
>>the upper namespace level upid nr is allowed to be predefined, since there
>>is not yet a way to take a snapshot of upid nrs at all namespaces levels.
>>
>>But this can easily be extended in the future.
>
>
> Good point, we will want to discuss the right way to dump that data. Do
> we add a new file under /proc/<pid>, use /proc/pid/status, or find some
> other way?
>
>
>>Signed-off-by: Nadia Derbey <Nadia.Derbey@bull.net>
>>
>>---
>> include/linux/next_syscall_data.h | 2
>> include/linux/pid.h | 2

```
>> kernel/fork.c                 |    3 -
>> kernel/pid.c                  |  111 ++++++++++++++++++++++++++++++++------
>> 4 files changed, 98 insertions(+), 20 deletions(-)
>>
>>Index: linux-2.6.26-rc5-mm3/kernel/pid.c
>>===================================================================
>>--- linux-2.6.26-rc5-mm3.orig/kernel/pid.c 2008-07-01 10:25:46.000000000 +0200
>>+++ linux-2.6.26-rc5-mm3/kernel/pid.c 2008-07-01 11:25:38.000000000 +0200
>>@@ -122,6 +122,26 @@ static void free_pidmap(struct upid *upi
>>  atomic_inc(&map->nr_free);
>> }
>>
>>+static inline int alloc_pidmap_page(struct pidmap *map)
>>+{
>>+ if (unlikely(!map->page)) {
>>+  void *page = kzalloc(PAGE_SIZE, GFP_KERNEL);
>>+  /*
>>+   * Free the page if someone raced with us
>>+   * installing it:
>>+   */
>>+  spin_lock_irq(&pidmap_lock);
>>+  if (map->page)
>>+   kfree(page);
>>+  else
>>+   map->page = page;
>>+  spin_unlock_irq(&pidmap_lock);
>>+  if (unlikely(!map->page))
>>+   return -1;
>>+ }
>>+ return 0;
>>+}
>>+
>> static int alloc_pidmap(struct pid_namespace *pid_ns)
>> {
>>  int i, offset, max_scan, pid, last = pid_ns->last_pid;
>>@@ -134,21 +154,8 @@ static int alloc_pidmap(struct pid_names
>>  map = &pid_ns->pidmap[pid/BITS_PER_PAGE];
>>  max_scan = (pid_max + BITS_PER_PAGE - 1)/BITS_PER_PAGE - !offset;
>>  for (i = 0; i <= max_scan; ++i) {
>>-  if (unlikely(!map->page)) {
>>-   void *page = kzalloc(PAGE_SIZE, GFP_KERNEL);
>>-   /*
>>-    * Free the page if someone raced with us
>>-    * installing it:
>>-    */
>>-   spin_lock_irq(&pidmap_lock);
>>-   if (map->page)
>>-    kfree(page);
```

```
>>-   else
>>-    map->page = page;
>>-   spin_unlock_irq(&pidmap_lock);
>>-   if (unlikely(!map->page))
>>-    break;
>>-  }
>>+  if (unlikely(alloc_pidmap_page(map)))
>>+   break;
>>   if (likely(atomic_read(&map->nr_free))) {
>>    do {
>>     if (!test_and_set_bit(offset, map->page)) {
>>@@ -182,6 +189,33 @@ static int alloc_pidmap(struct pid_names
>>  return -1;
>> }
>>
>>+/*
>>+ * Return 0 if successful (i.e. next_nr could be assigned as a upid nr).
>>+ * -errno else
>>+ */
>>+static int alloc_fixed_pidmap(struct pid_namespace *pid_ns, int next_nr)
>>+{
>>+ int offset;
>>+ struct pidmap *map;
>>+
>>+ if (next_nr < RESERVED_PIDS || next_nr >= pid_max)
>>+  return -EINVAL;
>>+
>>+ map = &pid_ns->pidmap[next_nr / BITS_PER_PAGE];
>>+
>>+ if (unlikely(alloc_pidmap_page(map)))
>>+  return -ENOMEM;
>>+
>>+ offset = next_nr & BITS_PER_PAGE_MASK;
>>+ if (test_and_set_bit(offset, map->page))
>>+  return -EBUSY;
>>+
>>+ atomic_dec(&map->nr_free);
>>+ pid_ns->last_pid = max(pid_ns->last_pid, next_nr);
>>+
>>+ return 0;
>>+}
>>+
>> int next_pidmap(struct pid_namespace *pid_ns, int last)
>> {
>>  int offset;
>>@@ -239,7 +273,25 @@ void free_pid(struct pid *pid)
>>  call_rcu(&pid->rcu, delayed_put_pid);
>> }
```

>>
>>-struct pid *alloc_pid(struct pid_namespace *ns)
>>+/*
>>+ * Sets a predefined upid nr for the process' upper namespace level
>>+ */
>>+static int set_predefined_pid(struct pid_namespace *ns, struct pid *pid,
>>+   int next_nr)
>>+{
>>+ int i = ns->level;
>>+ int rc;
>>+
>>+ rc = alloc_fixed_pidmap(ns, next_nr);
>>+ if (rc < 0)
>>+ return rc;
>>+
>>+ pid->numbers[i].nr = next_nr;
>>+ pid->numbers[i].ns = ns;
>>+ return 0;
>>+}
>>+
>>+struct pid *alloc_pid(struct pid_namespace *ns, int *retval)
>
>
> Is there a reason why you can't return retval using

No, except that I wanted that I wanted to change the fewest things. Now,
adding 1 argument vs adding 1 or 2 instructions, I don't mind: I'll
change that.

> return ERR_PTR(retval);
> instead of using an additional argument? Then at copy_process,
> after the call, do
>
> if (IS_ERR(pid))
>   retval = PTR_ERR(pid);
>
> ?
>
>
>> {
>> struct pid *pid;
>> enum pid_type type;
>>@@ -247,12 +299,37 @@ struct pid *alloc_pid(struct pid_namespa
>> struct pid_namespace *tmp;
>> struct upid *upid;
>>
>>+ *retval = -ENOMEM;
>> pid = kmem_cache_alloc(ns->pid_cachep, GFP_KERNEL);

```
>>  if (!pid)
>>   goto out;
>>
>>  tmp = ns;
>>- for (i = ns->level; i >= 0; i--) {
>>+ i = ns->level;
>>+ if (next_data_set(current)) {
>>+  /*
>>+   * There is a upid nr specified, use it instead of letting
>>+   * the kernel chose it for us.
>>+   */
>>+  int next_nr = get_next_data(current);
>>+  int rc;
>>+
>>+  rc = set_predefined_pid(tmp, pid, next_nr);
>>+  if (rc < 0) {
>>+   *retval = rc;
>>+   goto out_free;
>
>
> Again, I'd argue for resetting the syscall data on failure.
>
>
>>+  }
>>+  /* Go up one level */
>>+  tmp = tmp->parent;
>>+  i--;
>>+  reset_next_syscall_data(current);
>>+ }
>>+
>>+ /*
>>+  * Let the lower levels upid nrs be automatically allocated
>>+  */
>>+ *retval = -ENOMEM;
>>+ for ( ; i >= 0; i--) {
>>   nr = alloc_pidmap(tmp);
>>   if (nr < 0)
>>    goto out_free;
>>Index: linux-2.6.26-rc5-mm3/include/linux/pid.h
>>===================================================================
>>--- linux-2.6.26-rc5-mm3.orig/include/linux/pid.h 2008-07-01 10:25:46.000000000 +0200
>>+++ linux-2.6.26-rc5-mm3/include/linux/pid.h 2008-07-01 10:49:07.000000000 +0200
>>@@ -121,7 +121,7 @@ extern struct pid *find_get_pid(int nr);
>> extern struct pid *find_ge_pid(int nr, struct pid_namespace *);
>> int next_pidmap(struct pid_namespace *pid_ns, int last);
>>
>>-extern struct pid *alloc_pid(struct pid_namespace *ns);
>>+extern struct pid *alloc_pid(struct pid_namespace *, int *);
```

>> extern void free_pid(struct pid *pid);
>>
>> /*
>>Index: linux-2.6.26-rc5-mm3/kernel/fork.c
>>===
>>--- linux-2.6.26-rc5-mm3.orig/kernel/fork.c 2008-07-01 10:25:46.000000000 +0200
>>+++ linux-2.6.26-rc5-mm3/kernel/fork.c 2008-07-01 10:49:07.000000000 +0200
>>@@ -1110,8 +1110,7 @@ static struct task_struct *copy_process(
>> goto bad_fork_cleanup_io;
>>
>> if (pid != &init_struct_pid) {
>>- retval = -ENOMEM;
>>- pid = alloc_pid(task_active_pid_ns(p));
>>+ pid = alloc_pid(task_active_pid_ns(p), &retval);
>> if (!pid)
>>   goto bad_fork_cleanup_io;
>>
>>Index: linux-2.6.26-rc5-mm3/include/linux/next_syscall_data.h
>>===
>>--- linux-2.6.26-rc5-mm3.orig/include/linux/next_syscall_data.h 2008-07-01
10:41:36.000000000 +0200
>>+++ linux-2.6.26-rc5-mm3/include/linux/next_syscall_data.h 2008-07-01 11:09:35.000000000
+0200
>>@@ -5,6 +5,7 @@
>> * following is supported today:
>> *   . object creation with a predefined id
>> *   . for a sysv ipc object
>>+ *   . for a process
>> *
>> */
>>
>>@@ -19,6 +20,7 @@
>> * For example, it can be used to pre-set the id of the object to be created
>> * by next syscall. The following syscalls support this feature:
>> *   . msgget(), semget(), shmget()
>>+ *   . fork(), vfork(), clone()
>> */
>> struct next_syscall_data {
>> int ndata;
>>

Subject: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by Pavel Machek on Tue, 08 Jul 2008 10:51:43 GMT
View Forum Message <> Reply to Message

Hi!

>>> 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().
>>
>>
>> ...bloat task struct and
>>
>>
>>> 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.
>>
>>
>> ...introducing this kind of uglyness.
>>
>> Actually, there were proposals for sys_indirect(), which is slightly
>> less ugly, but IIRC we ended up with adding syscalls, too.

> I had a look at the lwn.net article that describes the sys_indirect()
> interface.
> It does exactly what we need here, so I do like it, but it has the same
> drawbacks as the one you're complaining about:
> . a new field is needed in the task structure
> . looks like many people found it ugly...

> Now, coming back to what I'm proposing: what we need is actually to change
> the behavior of *existing* syscalls, since we are in a very particular
> context (restarting an application).

Changing existing syscalls is _bad_: for backwards compatibility
reasons. strace will be very confusing to read, etc...

> Defining brand new syscalls is very touchy: needs to be careful about the
> interface + I can't imagine the number of syscalls that would be
> needed.

Of course new syscalls is touchy... modifying _existing_ should be
even more touchy.

Pavel

--
(english) http://www.livejournal.com/~pavelmachek
(cesky, pictures) http://atrey.karlin.mff.cuni.cz/~pavel/picture/horses/blog.html

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

---

Subject: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by Pavel Machek on Tue, 08 Jul 2008 10:52:28 GMT
View Forum Message <> Reply to Message

On Mon 2008-07-07 14:01:19, Serge E. Hallyn wrote:
> Quoting Pavel Machek (pavel@ucw.cz):
> > Hi!
> >
> > > 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.
> >
> > Yes, and I believe that's better than...
> >
> > > 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().
> >
> > ...bloat task struct and
> >
> > > 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.
> >
> > ...introducing this kind of uglyness.
> >
> > Actually, there were proposals for sys_indirect(), which is slightly
> > less ugly, but IIRC we ended up with adding syscalls, too.
>
> Silly question...
>
> Oren, would you object to defining sys_fork_with_id(),
> sys_msgget_with_id(), and sys_semget_with_id()?
>
> Eric, Pavel (Emelyanov), Dave, do you have preferences?
>
> For the cases Nadia has implemented here I'd be tempted to side with
> Pavel Machek, but once we get to things like open() and socket(), (a)
> the # new syscalls starts to jump, and (b) the per-syscall api starts to
> seem a lot more cumbersome.

You should not need to modify open/socket. You can already select fd
by creatively using open/dup/close...
	Pavel

--
(english) http://www.livejournal.com/~pavelmachek
(cesky, pictures) http://atrey.karlin.mff.cuni.cz/~pavel/picture/horses/blog.html

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

---

Subject: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by serue on Tue, 08 Jul 2008 21:47:21 GMT
View Forum Message <> Reply to Message

Quoting Pavel Machek (pavel@ucw.cz):
> Hi!
>
> >>> 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().
> >>
> >>
> >> ...bloat task struct and
> >>
> >>
> >>> 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.
> >>
> >>
> >> ...introducing this kind of uglyness.
> >>
> >> Actually, there were proposals for sys_indirect(), which is slightly
> >> less ugly, but IIRC we ended up with adding syscalls, too.
>
> > I had a look at the lwn.net article that describes the sys_indirect()
> > interface.
> > It does exactly what we need here, so I do like it, but it has the same
> > drawbacks as the one you're complaining about:
> > . a new field is needed in the task structure
> > . looks like many people found it ugly...
>
> > Now, coming back to what I'm proposing: what we need is actually to change
> > the behavior of *existing* syscalls, since we are in a very particular
> > context (restarting an application).
>
> Changing existing syscalls is _bad_: for backwards compatibility
> reasons. strace will be very confusing to read, etc...

I dunno... if you normally open(), you get back a random fd. If you do
it having set the next_id inadvertently, then as far as you know you get
back a random fd, no?

> > Defining brand new syscalls is very touchy: needs to be careful about the
> > interface + I can't imagine the number of syscalls that would be
> > needed.
>
> Of course new syscalls is touchy... modifying _existing_ should be
> even more touchy.

-serge

_____
Containers mailing list
Containers@lists.linux-foundation.org

Subject: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by serue on Tue, 08 Jul 2008 21:50:34 GMT
View Forum Message <> Reply to Message

Quoting Pavel Machek (pavel@ucw.cz):
> On Mon 2008-07-07 14:01:19, Serge E. Hallyn wrote:
> > Quoting Pavel Machek (pavel@ucw.cz):
> > > Hi!
> > >
> > > > 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.
> > >
> > > Yes, and I believe that's better than...
> > >
> > > > 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().
> > >
> > > ...bloat task struct and
> > >
> > > > 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.
> > >
> > > ...introducing this kind of uglyness.
> > >
> > > Actually, there were proposals for sys_indirect(), which is slightly
> > > less ugly, but IIRC we ended up with adding syscalls, too.
> >
> > Silly question...
> >
> > Oren, would you object to defining sys_fork_with_id(),
> > sys_msgget_with_id(), and sys_semget_with_id()?
> >
> > Eric, Pavel (Emelyanov), Dave, do you have preferences?
> >
> > For the cases Nadia has implemented here I'd be tempted to side with
> > Pavel Machek, but once we get to things like open() and socket(), (a)
> > the # new syscalls starts to jump, and (b) the per-syscall api starts to
> > seem a lot more cumbersome.
>
> You should not need to modify open/socket. You can already select fd
> by creatively using open/dup/close...

That's what we do right now in cryo. And if we end up patching up every API with separate syscalls, then we wouldn't create open_with_id(). But so long as the next_id were to exist, exploiting it in open is nigh on trivial and much nicer.

-serge

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

---

Subject: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by Pavel Machek on Tue, 08 Jul 2008 21:53:15 GMT
View Forum Message <> Reply to Message

On Tue 2008-07-08 16:47:21, Serge E. Hallyn wrote:
> Quoting Pavel Machek (pavel@ucw.cz):
> > Hi!
> >
> > >>> 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().
> > >>
> > >>
> > >> ...bloat task struct and
> > >>
> > >>
> > >>> 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.
> > >>
> > >>
> > >> ...introducing this kind of uglyness.
> > >>
> > >> Actually, there were proposals for sys_indirect(), which is slightly
> > >> less ugly, but IIRC we ended up with adding syscalls, too.
> >
> > > I had a look at the lwn.net article that describes the sys_indirect()
> > > interface.
> > > It does exactly what we need here, so I do like it, but it has the same
> > > drawbacks as the one you're complaining about:
> > > . a new field is needed in the task structure
> > > . looks like many people found it ugly...
> >
> > > Now, coming back to what I'm proposing: what we need is actually to change
> > > the behavior of *existing* syscalls, since we are in a very particular
> > > context (restarting an application).
> >
> > Changing existing syscalls is _bad_: for backwards compatibility
> > reasons. strace will be very confusing to read, etc...
>
> I dunno... if you normally open(), you get back a random fd. If you do
> it having set the next_id inadvertently, then as far as you know you get
> back a random fd, no?

Sorry?!

No, open does not return random fds. It allocates them bottom-up. So
you do not need any changes in open case.

(If you want to open "/foo/bar" as fd #50, open /dev/zero 49 times,
then open "/foo/bar"; bash already uses that trick.)
Pavel

--
(english) http://www.livejournal.com/~pavelmachek
(cesky, pictures) http://atrey.karlin.mff.cuni.cz/~pavel/picture/horses/blog.html

_____
Containers mailing list

Subject: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by Pavel Machek on Tue, 08 Jul 2008 21:58:21 GMT
View Forum Message <> Reply to Message

> > > > > 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().
> > > >
> > > > ...bloat task struct and
> > > >
> > > > > 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.
> > > >
> > > > ...introducing this kind of uglyness.
> > > >
> > > > Actually, there were proposals for sys_indirect(), which is slightly
> > > > less ugly, but IIRC we ended up with adding syscalls, too.
> > >
> > > Silly question...
> > >
> > > Oren, would you object to defining sys_fork_with_id(),
> > > sys_msgget_with_id(), and sys_semget_with_id()?
> > >
> > > Eric, Pavel (Emelyanov), Dave, do you have preferences?
> > >
> > > For the cases Nadia has implemented here I'd be tempted to side with
> > > Pavel Machek, but once we get to things like open() and socket(), (a)
> > > the # new syscalls starts to jump, and (b) the per-syscall api starts to
> > > seem a lot more cumbersome.
> >
> > You should not need to modify open/socket. You can already select fd
> > by creatively using open/dup/close...
>
> That's what we do right now in cryo. And if we end up patching up every
> API with separate syscalls, then we wouldn't create open_with_id(). But
> so long as the next_id were to exist, exploiting it in open is nigh on
> trivial and much nicer.

Ok, so ignore previous email. You know how unix works.

I believe you should just introduce syscalls you need. Yes,
introducing new syscalls is hard/expensive, but changing existing
syscalls is simply bad idea.

So what new syscalls do you _really_ need? Not open_this_fd, nor
socket_this_fd.

Pavel
--
(english) http://www.livejournal.com/~pavelmachek
(cesky, pictures) http://atrey.karlin.mff.cuni.cz/~pavel/picture/horses/blog.html

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

---

Subject: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by serue on Wed, 09 Jul 2008 02:20:35 GMT
View Forum Message <> Reply to Message

Quoting Pavel Machek (pavel@ucw.cz):
>
> > > > > > > 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().
> > > > > >
> > > > > > ...bloat task struct and
> > > > > >
> > > > > > > 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.
> > > > > >
> > > > > > ...introducing this kind of uglyness.
> > > > > >
> > > > > > Actually, there were proposals for sys_indirect(), which is slightly
> > > > > > less ugly, but IIRC we ended up with adding syscalls, too.
> > > > >
> > > > > Silly question...
> > > > >
> > > > > Oren, would you object to defining sys_fork_with_id(),
> > > > > sys_msgget_with_id(), and sys_semget_with_id()?
> > > > >
> > > > > Eric, Pavel (Emelyanov), Dave, do you have preferences?
> > > > >

> > > > For the cases Nadia has implemented here I'd be tempted to side with
> > > > Pavel Machek, but once we get to things like open() and socket(), (a)
> > > > the # new syscalls starts to jump, and (b) the per-syscall api starts to
> > > > seem a lot more cumbersome.
> > >
> > > You should not need to modify open/socket. You can already select fd
> > > by creatively using open/dup/close...
> >
> > That's what we do right now in cryo. And if we end up patching up every
> > API with separate syscalls, then we wouldn't create open_with_id(). But
> > so long as the next_id were to exist, exploiting it in open is nigh on
> > trivial and much nicer.
>
> Ok, so ignore previous email. You know how unix works.
>
> I believe you should just introduce syscalls you need. Yes,
> introducing new syscalls is hard/expensive, but changing existing
> syscalls is simply bad idea.

Ok, thanks, Pavel. I'm really far more inclined to agree with you than
it probably sounds like. I'll go ahead and implement a clone_with_id()
syscall for starters later this week just as a comparison.

Unless, Nadia, you have already started that?

> So what new syscalls do you _really_ need? Not open_this_fd, nor
> socket_this_fd.

Oren, do you have a list of the syscalls which were modified to use the
next_id in zap?

-serge

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

---

Subject: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by Nadia Derbey on Thu, 10 Jul 2008 06:54:10 GMT
View Forum Message <> Reply to Message

Pavel Machek wrote:
> On Tue 2008-07-08 16:47:21, Serge E. Hallyn wrote:
>
>>Quoting Pavel Machek (pavel@ucw.cz):
>>
>>>Hi!

>>>
>>>
>>>>>>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().
>>>>>
>>>>>
>>>>>...bloat task struct and
>>>>>
>>>>>
>>>>>
>>>>>>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.
>>>>>
>>>>>
>>>>>...introducing this kind of uglyness.
>>>>>
>>>>>Actually, there were proposals for sys_indirect(), which is slightly
>>>>>less ugly, but IIRC we ended up with adding syscalls, too.
>>>
>>>>I had a look at the lwn.net article that describes the sys_indirect()
>>>>interface.
>>>>It does exactly what we need here, so I do like it, but it has the same
>>>>drawbacks as the one you're complaining about:
>>>>. a new field is needed in the task structure
>>>>. looks like many people found it ugly...
>>>
>>>>Now, coming back to what I'm proposing: what we need is actually to change
>>>>the behavior of *existing* syscalls, since we are in a very particular
>>>>context (restarting an application).
>>>
>>>Changing existing syscalls is _bad_: for backwards compatibility
>>>reasons. strace will be very confusing to read, etc...
>>
>>I dunno... if you normally open(), you get back a random fd. If you do
>>it having set the next_id inadvertently, then as far as you know you get
>>back a random fd, no?
>
>
> Sorry?!
>
> No, open does not return random fds. It allocates them bottom-up. So
> you do not need any changes in open case.
>

> (If you want to open "/foo/bar" as fd #50, open /dev/zero 49 times,

49 times - <# of already busy fds>

Don't you think it's simpler to specify the target fd, and then open the
file.

> then open "/foo/bar"; bash already uses that trick.)
> Pavel
>

Regards,
Nadia

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

---

Subject: Re: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by Paul Menage on Thu, 10 Jul 2008 07:01:55 GMT
View Forum Message <> Reply to Message

On Wed, Jul 9, 2008 at 11:54 PM, Nadia Derbey <Nadia.Derbey@bull.net> wrote:
>
> Don't you think it's simpler to specify the target fd, and then open the
> file.

Maybe. But:

- this can already be done without extra kernel support, via open()
followed by dup2()

- if you were going to add it to the kernel, the precedent set by
openat() is that you create a new system call that supports the
extended semantics.

Paul

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

---

Subject: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by Nadia Derbey on Thu, 10 Jul 2008 07:42:03 GMT
View Forum Message <> Reply to Message

Pavel Machek wrote:
> Hi!
>
>
>>>>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().
>>>
>>>
>>>...bloat task struct and
>>>
>>>
>>>
>>>>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.
>>>
>>>
>>>...introducing this kind of uglyness.
>>>
>>>Actually, there were proposals for sys_indirect(), which is slightly
>>>less ugly, but IIRC we ended up with adding syscalls, too.
>
>
>>I had a look at the lwn.net article that describes the sys_indirect()
>>interface.
>>It does exactly what we need here, so I do like it, but it has the same
>>drawbacks as the one you're complaining about:
>>. a new field is needed in the task structure
>>. looks like many people found it ugly...
>
>
>>Now, coming back to what I'm proposing: what we need is actually to change
>>the behavior of *existing* syscalls, since we are in a very particular
>>context (restarting an application).
>
>
> Changing existing syscalls is _bad_: for backwards compatibility
> reasons.

I'm sorry but I don't see a backward compatibility problem: same
interface, same functionality provided. The only change is in the way

ids are assigned.

Actually, one drawback I'm seeing is that we are adding a test to the classical syscall path (the test on the current->next_syscall_data being set or not).

> strace will be very confusing to read, etc...

We'll have the 3 following lines added to an strace output each time we fill the proc file:

```
open("/proc/15084/task/15084/next_syscall_data", O_RDWR) = 4
write(4, "LONG1 100", 9)              = 9
close(4)                              = 0
```

I don't see anthing confusing here ;-)

Regards,
Nadia

>
>
>>Defining brand new syscalls is very touchy: needs to be careful about the
>>interface + I can't imagine the number of syscalls that would be
>>needed.
>
>
> Of course new syscalls is touchy... modifying _existing_ should be
> even more touchy.
>
> Pavel
>

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

---

Subject: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by Nadia Derbey on Thu, 10 Jul 2008 07:58:48 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Pavel Machek (pavel@ucw.cz):
>

>>>>>>>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().
>>>>>>
>>>>>>...bloat task struct and
>>>>>>
>>>>>>
>>>>>>>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.
>>>>>>
>>>>>>...introducing this kind of uglyness.
>>>>>>
>>>>>>Actually, there were proposals for sys_indirect(), which is slightly
>>>>>>less ugly, but IIRC we ended up with adding syscalls, too.
>>>>>
>>>>>Silly question...
>>>>>
>>>>>Oren, would you object to defining sys_fork_with_id(),
>>>>>sys_msgget_with_id(), and sys_semget_with_id()?
>>>>>
>>>>>Eric, Pavel (Emelyanov), Dave, do you have preferences?
>>>>>
>>>>>For the cases Nadia has implemented here I'd be tempted to side with
>>>>>Pavel Machek, but once we get to things like open() and socket(), (a)
>>>>>the # new syscalls starts to jump, and (b) the per-syscall api starts to
>>>>>seem a lot more cumbersome.
>>>>
>>>>You should not need to modify open/socket. You can already select fd
>>>>by creatively using open/dup/close...
>>>
>>>That's what we do right now in cryo. And if we end up patching up every
>>>API with separate syscalls, then we wouldn't create open_with_id(). But
>>>so long as the next_id were to exist, exploiting it in open is nigh on
>>>trivial and much nicer.
>>
>>Ok, so ignore previous email. You know how unix works.
>>
>>I believe you should just introduce syscalls you need. Yes,
>>introducing new syscalls is hard/expensive, but changing existing
>>syscalls is simply bad idea.
>
>
> Ok, thanks, Pavel. I'm really far more inclined to agree with you than
> it probably sounds like. I'll go ahead and implement a clone_with_id()

> syscall for starters later this week just as a comparison.
>
> Unless, Nadia, you have already started that?

Actually, what I've started working on these days is replace the proc
interface by a syscall to set the next_syscall_data field: I think this
might help us avoid defining a precise list of the new syscalls we need?

Regards,
Nadia

>
>
>>So what new syscalls do you _really_ need? Not open_this_fd, nor
>>socket_this_fd.
>
>
> Oren, do you have a list of the syscalls which were modified to use the
> next_id in zap?
>
> -serge
>
>

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

---

Subject: Re: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by Paul Menage on Thu, 10 Jul 2008 08:34:32 GMT
View Forum Message <> Reply to Message

On Thu, Jul 10, 2008 at 12:58 AM, Nadia Derbey <Nadia.Derbey@bull.net> wrote:
>
> Actually, what I've started working on these days is replace the proc
> interface by a syscall to set the next_syscall_data field: I think this
> might help us avoid defining a precise list of the new syscalls we need?

Isn't that just sys_indirect(), but split into two syscall invocations
rather than one?

Paul

_____
Containers mailing list
Containers@lists.linux-foundation.org

Subject: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by Pavel Machek on Thu, 10 Jul 2008 08:54:06 GMT
View Forum Message <> Reply to Message

On Thu 2008-07-10 09:42:03, Nadia Derbey wrote:
> Pavel Machek wrote:
>> Hi!
>>
>>
>>>>>> 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().
>>>>
>>>>
>>>> ...bloat task struct and
>>>>
>>>>
>>>>
>>>>>> 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.
>>>>
>>>>
>>>> ...introducing this kind of uglyness.
>>>>
>>>> Actually, there were proposals for sys_indirect(), which is slightly
>>>> less ugly, but IIRC we ended up with adding syscalls, too.
>>
>>
>>> I had a look at the lwn.net article that describes the sys_indirect()
>>> interface.
>>> It does exactly what we need here, so I do like it, but it has the same
>>> drawbacks as the one you're complaining about:
>>> . a new field is needed in the task structure
>>> . looks like many people found it ugly...
>>
>>
>>> Now, coming back to what I'm proposing: what we need is actually to
>>> change the behavior of *existing* syscalls, since we are in a very
>>> particular context (restarting an application).
>>
>>

>> Changing existing syscalls is _bad_: for backwards compatibility
>> reasons.
>
> I'm sorry but I don't see a backward compatibility problem: same interface,
> same functionality provided. The only change is in the way ids are
> assigned.

If you don't see a backward compatibility problem here, perhaps you
should not be hacking kernel...? The way ids are assigned is certainly
part of syscall semantics (applications rely on), at least for open.

If you want to claim that your solution is better than adding milion
of syscalls, I guess you need to list the milion of syscalls, so we
can compare.

> Actually, one drawback I'm seeing is that we are adding a test to the
> classical syscall path (the test on the current->next_syscall_data being
> set or not).
>
>> strace will be very confusing to read, etc...
>
> We'll have the 3 following lines added to an strace output each time we
> fill the proc file:
>
> open("/proc/15084/task/15084/next_syscall_data", O_RDWR) = 4
> write(4, "LONG1 100", 9)   = 9
> close(4)   = 0
>
> I don't see anthing confusing here ;-)

No, that part is just very very ugly.

close(5)
close(6)
open("foo") = 6

_is_ confusing to me.
        Pavel
--
(english) http://www.livejournal.com/~pavelmachek
(cesky, pictures) http://atrey.karlin.mff.cuni.cz/~pavel/picture/horses/blog.html
_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by Nadia Derbey on Thu, 10 Jul 2008 09:14:18 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On Wed, Jul 9, 2008 at 11:54 PM, Nadia Derbey <Nadia.Derbey@bull.net> wrote:
>
>>Don't you think it's simpler to specify the target fd, and then open the
>>file.
>
>
> Maybe. But:
>
> - this can already be done without extra kernel support, via open()
> followed by dup2()

Sure, I completely agree with you.
Actually, that's the way it is handled in cryo code.
But I think that both ways of doing are not exactly the same in case of
failure:
open + dup2 will close newfd if it is already busy.
while
next-syscall_data + open will fail if the target fd is already busy. And
that's the functionality we need during restart, isn't it?

Regards,
Nadia

>
> - if you were going to add it to the kernel, the precedent set by
> openat() is that you create a new system call that supports the
> extended semantics.
>
>

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

Subject: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by Nadia Derbey on Thu, 10 Jul 2008 09:29:45 GMT
View Forum Message <> Reply to Message

Pavel Machek wrote:
> On Thu 2008-07-10 09:42:03, Nadia Derbey wrote:
>
>>Pavel Machek wrote:
>>
>>>Hi!
>>>
>>>
>>>
>>>>>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().
>>>>>
>>>>>
>>>>>...bloat task struct and
>>>>>
>>>>>
>>>>>
>>>>>
>>>>>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.
>>>>>
>>>>>
>>>>>...introducing this kind of uglyness.
>>>>>
>>>>>Actually, there were proposals for sys_indirect(), which is slightly
>>>>>less ugly, but IIRC we ended up with adding syscalls, too.
>>>
>>>
>>>>I had a look at the lwn.net article that describes the sys_indirect()
>>>>interface.
>>>>It does exactly what we need here, so I do like it, but it has the same
>>>>drawbacks as the one you're complaining about:
>>>>. a new field is needed in the task structure
>>>>. looks like many people found it ugly...
>>>
>>>
>>>>Now, coming back to what I'm proposing: what we need is actually to
>>>>change the behavior of *existing* syscalls, since we are in a very
>>>>particular context (restarting an application).
>>>
>>>
>>>Changing existing syscalls is _bad_: for backwards compatibility
>>>reasons.

>>
>>I'm sorry but I don't see a backward compatibility problem: same interface,
>>same functionality provided. The only change is in the way ids are
>>assigned.
>
>
> If you don't see a backward compatibility problem here, perhaps you
> should not be hacking kernel...?

Thx for the advice, will try think about it...

> The way ids are assigned is certainly
> part of syscall semantics (applications rely on), at least for open.
>
> If you want to claim that your solution is better than adding milion
> of syscalls, I guess you need to list the milion of syscalls, so we
> can compare.
>

I'm not claiming anything: just trying to see what actually are the
pro's and con's for any proposed solution.

Regards,
Nadia

>
>>Actually, one drawback I'm seeing is that we are adding a test to the
>>classical syscall path (the test on the current->next_syscall_data being
>>set or not).
>>
>>
>>>strace will be very confusing to read, etc...
>>
>>We'll have the 3 following lines added to an strace output each time we
>>fill the proc file:
>>
>>open("/proc/15084/task/15084/next_syscall_data", O_RDWR) = 4
>>write(4, "LONG1 100", 9)   = 9
>>close(4)   = 0
>>
>>I don't see anthing confusing here ;-)
>
>
> No, that part is just very very ugly.
>
> close(5)
> close(6)
> open("foo") = 6

>
> _is_ confusing to me.
> Pavel

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

---

Subject: Re: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by Paul Menage on Thu, 10 Jul 2008 09:30:26 GMT
View Forum Message <> Reply to Message

On Thu, Jul 10, 2008 at 2:14 AM, Nadia Derbey <Nadia.Derbey@bull.net> wrote:
> But I think that both ways of doing are not exactly the same in case of
> failure:
> open + dup2 will close newfd if it is already busy.
> while
> next-syscall_data + open will fail if the target fd is already busy. And
> that's the functionality we need during restart, isn't it?

No, I don't think so. The cryo restart code should be aware of exactly
which fds it has open and which it needs to open, and can shuffle them
around as necessary via dup2() to get them into the right places.

Paul

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

---

Subject: Re: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by Nadia Derbey on Thu, 10 Jul 2008 09:38:45 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On Thu, Jul 10, 2008 at 12:58 AM, Nadia Derbey <Nadia.Derbey@bull.net> wrote:
>
>>Actually, what I've started working on these days is replace the proc
>>interface by a syscall to set the next_syscall_data field: I think this
>>might help us avoid defining a precise list of the new syscalls we need?
>
>

> Isn't that just sys_indirect(), but split into two syscall invocations
> rather than one?
>

Yes, from what I've read about the sys_indirect(), it is.
Unfortunalty, I hadn't followed the thread, so except because of its
"ugliness" (again ;-)) I don't exactly know why the idea has been given up.

Regards,
Nadia

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

---

Subject: Re: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by Nadia Derbey on Thu, 10 Jul 2008 10:11:20 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On Thu, Jul 10, 2008 at 2:14 AM, Nadia Derbey <Nadia.Derbey@bull.net> wrote:
>
>>But I think that both ways of doing are not exactly the same in case of
>>failure:
>>open + dup2 will close newfd if it is already busy.
>>while
>>next-syscall_data + open will fail if the target fd is already busy. And
>>that's the functionality we need during restart, isn't it?
>
>
> No, I don't think so. The cryo restart code should be aware of exactly
> which fds it has open and which it needs to open, and can shuffle them
> around as necessary via dup2() to get them into the right places.
>

Yes sure, that's exactly what it does.

what I just wanted to say here is that the concept of opening + dup2'ing
is not exactly the same as the open_with_id concept.

But I agree with you: a restart code knows exactly what it does and can
safely play with the open and dup syscalls.

Regards,
Nadia

_____

Subject: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by Dave Hansen on Thu, 10 Jul 2008 17:53:35 GMT
View Forum Message <> Reply to Message

On Thu, 2008-07-10 at 10:54 +0200, Pavel Machek wrote:
>
> If you don't see a backward compatibility problem here, perhaps you
> should not be hacking kernel...? The way ids are assigned is certainly
> part of syscall semantics (applications rely on), at least for open.

We also used to have a pretty defined ordering for handing out address space with mmap(). That all changed with address space randomization. Are file descriptors different somehow?

Anyway, it's not like we're actually changing existing behavior. An application has to do something special and new to trigger this new behavior. Nobody is going to stumble over it, and it will *not* break backward compatibility.

-- Dave

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

Subject: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by Pavel Machek on Thu, 10 Jul 2008 18:45:12 GMT
View Forum Message <> Reply to Message

On Thu 2008-07-10 10:53:35, Dave Hansen wrote:
> On Thu, 2008-07-10 at 10:54 +0200, Pavel Machek wrote:
> >
> > If you don't see a backward compatibility problem here, perhaps you
> > should not be hacking kernel...? The way ids are assigned is certainly
> > part of syscall semantics (applications rely on), at least for open.
>
> We also used to have a pretty defined ordering for handing out address
> space with mmap(). That all changed with address space randomization.
> Are file descriptors different somehow?
>

> Anyway, it's not like we're actually changing existing behavior. An
> application has to do something special and new to trigger this new
> behavior. Nobody is going to stumble over it, and it will *not* break
> backward compatibility.

It will break compatibility, but not in a way you expect. There's application called "subterfugue" that monitors other applications using ptrace and enforces security policy (or does other stuff). Such hacks depend on existing syscalls behaving in a way they are specified...

Then you'll have to update open.2 man page:

DESCRIPTION
       Given a pathname for a file, open() returns a file descriptor, a small, non-
       negative integer for use in subsequent system calls (read(2), write(2),
       lseek(2), fcntl(2), etc.).   The file descriptor returned by a successful
       call will be the lowest-numbered file descriptor not currently open for the
       process.

...you'll need to add "unless someone write some number in file in /proc somewhere"... hmm... is new behaviour even POSIX compliant? open() is specified in POSIX...

Ok, so it will not break too many apps... but echo "123 > /proc/something" breaking bash (etc) is not nice.

(Plus proposed interface is so ugly that this discussion is moot.)

        Pavel
--
(english) http://www.livejournal.com/~pavelmachek
(cesky, pictures) http://atrey.karlin.mff.cuni.cz/~pavel/picture/horses/blog.html

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

---

Subject: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by Dave Hansen on Thu, 10 Jul 2008 19:04:03 GMT
View Forum Message <> Reply to Message

On Thu, 2008-07-10 at 20:45 +0200, Pavel Machek wrote:

> On Thu 2008-07-10 10:53:35, Dave Hansen wrote:
> > On Thu, 2008-07-10 at 10:54 +0200, Pavel Machek wrote:
> > >
> > > If you don't see a backward compatibility problem here, perhaps you
> > > should not be hacking kernel...? The way ids are assigned is certainly
> > > part of syscall semantics (applications rely on), at least for open.
> >
> > We also used to have a pretty defined ordering for handing out address
> > space with mmap(). That all changed with address space randomization.
> > Are file descriptors different somehow?
> >
> > Anyway, it's not like we're actually changing existing behavior. An
> > application has to do something special and new to trigger this new
> > behavior. Nobody is going to stumble over it, and it will *not* break
> > backward compatibility.
>
> It will break compatibility, but not in a way you expect. There's
> application called "subterfugue" that monitors other applications
> using ptrace and enforces security policy (or does other stuff). Such
> hacks depend on existing syscalls behaving in a way they are
> specified...
>
> Then you'll have to update open.2 man page:
>
> DESCRIPTION
> Given a pathname for a file, open() returns a file descriptor,
> a small, non-
> negative integer for use in subsequent system calls
> (read(2), write(2),
> lseek(2), fcntl(2), etc.). The file descriptor returned by
> a successful
> call will be the lowest-numbered file descriptor not currently
> open for the
> process.
>
> ...you'll need to add "unless someone write some number in file in
> /proc somewhere"... hmm... is new behaviour even POSIX compliant?
> open() is specified in POSIX...

Yup, that's true. Good point.

> Ok, so it will not break too many apps... but echo "123 >
> /proc/something" breaking bash (etc) is not nice.
>
> (Plus proposed interface is so ugly that this discussion is moot.)

Yes, I agree that the current proposed interface is too ugly to live. :)

-- Dave

_____

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

---

Subject: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by serue on Thu, 10 Jul 2008 19:27:20 GMT
View Forum Message <> Reply to Message

Quoting Dave Hansen (dave@linux.vnet.ibm.com):
> On Thu, 2008-07-10 at 20:45 +0200, Pavel Machek wrote:
> > On Thu 2008-07-10 10:53:35, Dave Hansen wrote:
> > > On Thu, 2008-07-10 at 10:54 +0200, Pavel Machek wrote:
> > > >
> > > > If you don't see a backward compatibility problem here, perhaps you
> > > > should not be hacking kernel...? The way ids are assigned is certainly
> > > > part of syscall semantics (applications rely on), at least for open.
> > >
> > > We also used to have a pretty defined ordering for handing out address
> > > space with mmap(). That all changed with address space randomization.
> > > Are file descriptors different somehow?
> > >
> > > Anyway, it's not like we're actually changing existing behavior. An
> > > application has to do something special and new to trigger this new
> > > behavior. Nobody is going to stumble over it, and it will *not* break
> > > backward compatibility.
> >
> > It will break compatibility, but not in a way you expect. There's
> > application called "subterfugue" that monitors other applications
> > using ptrace and enforces security policy (or does other stuff). Such
> > hacks depend on existing syscalls behaving in a way they are
> > specified...
> >
> > Then you'll have to update open.2 man page:
> >
> > DESCRIPTION
> > Given a pathname for a file, open() returns a file descriptor,
> > a small, non-
> > negative integer for use in subsequent system calls
> > (read(2), write(2),
> > lseek(2), fcntl(2), etc.). The file descriptor returned by
> > a successful
> > call will be the lowest-numbered file descriptor not currently
> > open for the
> > process.

> >
> > ...you'll need to add "unless someone write some number in file in
> > /proc somewhere"... hmm... is new behaviour even POSIX compliant?
> > open() is specified in POSIX...
>
> Yup, that's true. Good point.

I didn't think it was, as I thought it was current behavior but not
mandated by the spec.

But I was wrong.

So this patch must be dropped, at any rate.

> > Ok, so it will not break too many apps... but echo "123 >
> > /proc/something" breaking bash (etc) is not nice.
> >
> > (Plus proposed interface is so ugly that this discussion is moot.)
>
> Yes, I agree that the current proposed interface is too ugly to live. :)

-serge

_____

Subject: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by Oren Laadan on Thu, 17 Jul 2008 22:26:43 GMT
View Forum Message <> Reply to Message

Wow ... the volume of messages in this thread is overwhelming.
I guess it had to happen when I was off a month long vacation !

Ok .. lemme see if I can catch up:

Serge E. Hallyn wrote:
> Quoting Pavel Machek (pavel@ucw.cz):
>>>>>>> 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().
>>>>>> ...bloat task struct and
>>>>>>
>>>>>>> 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.
>>>>>> ...introducing this kind of uglyness.
>>>>>>
>>>>>> Actually, there were proposals for sys_indirect(), which is slightly
>>>>>> less ugly, but IIRC we ended up with adding syscalls, too.
>>>>> Silly question...
>>>>>
>>>>> Oren, would you object to defining sys_fork_with_id(),
>>>>> sys_msgget_with_id(), and sys_semget_with_id()?

I don't object, in particular given the backward-compatibility issue
that was discussed later in this thread. However, see more below.

>>>>>
>>>>> Eric, Pavel (Emelyanov), Dave, do you have preferences?
>>>>>
>>>>> For the cases Nadia has implemented here I'd be tempted to side with
>>>>> Pavel Machek, but once we get to things like open() and socket(), (a)
>>>>> the # new syscalls starts to jump, and (b) the per-syscall api starts to
>>>>> seem a lot more cumbersome.
>>>> You should not need to modify open/socket. You can already select fd
>>>> by creatively using open/dup/close...
>>> That's what we do right now in cryo. And if we end up patching up every
>>> API with separate syscalls, then we wouldn't create open_with_id(). But
>>> so long as the next_id were to exist, exploiting it in open is nigh on
>>> trivial and much nicer.
>> Ok, so ignore previous email. You know how unix works.
>>
>> I believe you should just introduce syscalls you need. Yes,
>> introducing new syscalls is hard/expensive, but changing existing
>> syscalls is simply bad idea.
>
> Ok, thanks, Pavel. I'm really far more inclined to agree with you than
> it probably sounds like. I'll go ahead and implement a clone_with_id()
> syscall for starters later this week just as a comparison.
>
> Unless, Nadia, you have already started that?
>
>> So what new syscalls do you _really_ need? Not open_this_fd, nor
>> socket_this_fd.
>
> Oren, do you have a list of the syscalls which were modified to use the
> next_id in zap?

Good question :)

In zap, all of the checkpoint, and most of the restart is performed in

kernel space. The user space component of the restart takes care of the creation of the process tree correctly with the desired SID for each process. Thus, the _only_ syscall that requires this hack from userland is clone(). I'm ok with adding a new syscall to do this job.

Everything else is created from within the kernel, usually by invoking the appropriate syscall inside. I use a similar trick (but in this case, only visible from within the kernel, not settable from user space, so there is never an issue with backward compatibility) for SysV IPC (select virtual ID) and PTY (select slave number when opening /dev/ptmx).

As mentioned by others, FD's are adjusted with dup2() if necessary.

Lastly, I can envision a need for a similar trick with certain devices if they are to be supported (e.g., if /dev/rtc is modified to work per namespace etc). But I wouldn't bother about that now.

Oren.