Subject: [PATCH 0/4] - v2 - Object creation with a specified id
Posted by Nadia Derbey on Fri, 18 Apr 2008 05:44:59 GMT
View Forum Message <> Reply to Message

When restarting a process that has been previously checkpointed, that process should keep on using some of its ids (such as its process id, or sysV ipc ids).

This patch provides a feature that can help ensuring this saved state reuse: it makes it possible to create an object with a pre-defined id.

A first implementation had been proposed 2 months ago. It consisted in changing an object's id after it had been created.

Here is a second implementation based on Oren Ladaan's idea: Oren's suggestion was to force an object's id during its creation, rather than 1. create it, 2. change its id.

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

When this file is filled with an id value, a structure pointed to by the calling task_struct is filled with that id.

Then, when an object supporting this feature is created, the id present in that new structure is used, instead of the default one.

The syntax is one of:
. echo "LONG XX" > /proc/self/task/<my_tid>/next_id
  next object to be created will have an id set to XX
. echo "LONG<n> X0 ... X<n-1>" > /proc/self/task/<my_tid>/next_id
  next object to be created will have its ids set to XX0, ... X<n-1>
  This is particularly useful for processes that may have several ids if they belong to nested namespaces.

The objects covered here are ipc objects and processes.
Today, the ids are specified as long, but having a type string specified in the next_id file makes it possible to cover more types in the future, if needed.

The patches are against 2.6.25-rc3-mm2, in the following order:

[PATCH 1/4] adds the procfs facility for next object to be created, this object being associated to a single id.
[PATCH 2/4] enhances the procfs facility for objects associated to multiple ids (like processes).
[PATCH 3/4] makes use of the specified id (if any) to allocate the new IPC object (changes the ipc_addid() path).

[PATCH 4/4] uses the specified id(s) (if any) to set the upid nr(s) for a newly
allocated process (changes the alloc_pid()/alloc_pidmap() paths).

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: [PATCH 4/4] - v2 - PID: use the target ID specified in procfs
Posted by Nadia Derbey on Fri, 18 Apr 2008 05:45:03 GMT
View Forum Message <> Reply to Message

[PATCH 04/04]

This patch makes use of the target ids specified by a previous write to
/proc/self/task/<tid>/next_id as the ids to use to allocate the next upid nrs.
Upper levels upid nrs that are not specified in next_pids file are left to the
kernel choice.

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

```
---
 include/linux/pid.h |    2
 kernel/fork.c       |    3 -
 kernel/pid.c        |  141 ++++++++++++++++++++++++++++++++++++++++++++++-------
 3 files changed, 126 insertions(+), 20 deletions(-)

Index: linux-2.6.25-rc8-mm2/include/linux/pid.h
===================================================================
--- linux-2.6.25-rc8-mm2.orig/include/linux/pid.h 2008-04-17 12:50:22.000000000 +0200
+++ linux-2.6.25-rc8-mm2/include/linux/pid.h 2008-04-17 17:42:11.000000000 +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 *ns, int *retval);
 extern void free_pid(struct pid *pid);

 /*
Index: linux-2.6.25-rc8-mm2/kernel/fork.c
```

```
===================================================================
--- linux-2.6.25-rc8-mm2.orig/kernel/fork.c 2008-04-17 13:49:09.000000000 +0200
+++ linux-2.6.25-rc8-mm2/kernel/fork.c 2008-04-17 17:42:48.000000000 +0200
@@ -1200,8 +1200,7 @@ 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-2.6.25-rc8-mm2/kernel/pid.c
===================================================================
--- linux-2.6.25-rc8-mm2.orig/kernel/pid.c 2008-04-17 12:50:25.000000000 +0200
+++ linux-2.6.25-rc8-mm2/kernel/pid.c 2008-04-17 17:50:00.000000000 +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 +189,35 @@ static int alloc_pidmap(struct pid_names
  return -1;
 }

+/*
+ * Return a predefined pid value if successful (ID_AT(pid_l, level)),
+ * -errno else
+ */
+static int alloc_fixed_pidmap(struct pid_namespace *pid_ns,
+   struct sys_id *pid_l, int level)
+{
+ int offset, pid;
+ struct pidmap *map;
+
+ pid = ID_AT(pid_l, level);
+ if (pid < RESERVED_PIDS || pid >= pid_max)
+  return -EINVAL;
+
+ map = &pid_ns->pidmap[pid / BITS_PER_PAGE];
+
+ if (unlikely(alloc_pidmap_page(map)))
+  return -ENOMEM;
+
+ offset = pid & 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, pid);
+
+ return pid;
+}
+
 int next_pidmap(struct pid_namespace *pid_ns, int last)
 {
  int offset;
@@ -243,20 +279,91 @@ void free_pid(struct pid *pid)
  call_rcu(&pid->rcu, delayed_put_pid);
 }

-struct pid *alloc_pid(struct pid_namespace *ns)
+/*
+ * Called by alloc_pid() to use a list of predefined ids for the calling
+ * process' upper ns levels.
+ * Returns next pid ns to visit if successful (may be NULL if walked through
+ * the entire pid ns hierarchy).
+ * i is filled with next level to be visited (useful for the error cases).
+ */
+static struct pid_namespace *set_predefined_pids(struct pid_namespace *ns,
+       struct pid *pid,
+       struct sys_id *pid_l,
+       int *next_level)
+{
+ struct pid_namespace *tmp;
+ int rel_level, i, nr;
+
+ rel_level = pid_l->nids - 1;
+ if (rel_level > ns->level)
+  return ERR_PTR(-EINVAL);
+
+ tmp = ns;
+
+ /*
+  * Use the predefined upid nrs for levels ns->level down to
+  * ns->level - rel_level
+  */
+ for (i = ns->level ; rel_level >= 0; i--, rel_level--) {
+  nr = alloc_fixed_pidmap(tmp, pid_l, rel_level);
+  if (nr < 0) {
+   tmp = ERR_PTR(nr);
+   goto out;
+  }
+
+  pid->numbers[i].nr = nr;
+  pid->numbers[i].ns = tmp;
+  tmp = tmp->parent;
```

```
+ }
+
+ id_blocks_free(pid_l);
+out:
+ *next_level = i;
+ return tmp;
+}
+
+struct pid *alloc_pid(struct pid_namespace *ns, int *retval)
 {
  struct pid *pid;
  enum pid_type type;
  int i, nr;
  struct pid_namespace *tmp;
  struct upid *upid;
+ struct sys_id *pid_l;

+ *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 there is a list of upid nrs specified, use it instead of letting
+  * the kernel chose the upid nrs for us.
+  */
+ pid_l = current->next_id;
+ if (pid_l && pid_l->nids) {
+  /*
+   * returns the next ns to be visited in the following loop
+   * (or NULL if we are done).
+   * i is filled in with the next level to be visited. We need
+   * it to undo things in the error cases.
+   */
+  tmp = set_predefined_pids(ns, pid, pid_l, &i);
+  if (IS_ERR(tmp)) {
+   *retval = PTR_ERR(tmp);
+   goto out_free;
+  }
+  current->next_id = NULL;
+ }
+
+ *retval = -ENOMEM;
+ /*
```

```
+  * Let the lower levels upid nrs be automatically allocated
+  */
+ for ( ; i >= 0; i--) {
   nr = alloc_pidmap(tmp);
   if (nr < 0)
    goto out_free;

--
```

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

---

Subject: Re: [PATCH 0/4] - v2 - Object creation with a specified id
Posted by Dave Hansen on Fri, 18 Apr 2008 17:07:07 GMT
View Forum Message <> Reply to Message

On Fri, 2008-04-18 at 07:44 +0200, Nadia.Derbey@bull.net wrote:
> The syntax is one of:
>   . echo "LONG XX" > /proc/self/task/<my_tid>/next_id
>   next object to be created will have an id set to XX
>   . echo "LONG<n> X0 ... X<n-1>" > /proc/self/task/<my_tid>/next_id
>   next object to be created will have its ids set to XX0, ... X<n-1>
>   This is particularly useful for processes that may have several ids if
>   they belong to nested namespaces.

I still think, in its current form, this is pretty dangerous.

It might be nice to have a central place to set ids, but there's no real safety here for when we get a mishmash of 50 of these things. We have a queue and we fix the order in which things are inserted, but how do we control the order in which the kernel extracts them? Can we 100% conform to that order in the future and be consistent? Won't that also be dictated by the way userspace behaves?

If we're going to go this route, I think we need to do a few things. First, these need to be more "type safe". That means that sticking an ids in here that you mean to go to an IPC should never, ever, ever be able to get interpreted as a PID or as an inode number or anything else.

```
 echo "IPC_SEM_ID NN" > /proc/self/task/<my_tid>/next_id
 echo "PID MM" > /proc/self/task/<my_tid>/next_id
```

Maybe we should include the types in there as well, but those are kinda implied by the kind of id you're setting. Have you thought about 32-bit userspace with 64-bit kernels? Does that cause any issues?

I also feel like we should have the kernel able to enumerate which
things in a process *are* settable. What are the valid values for that
first word being echoed in the file?

Does anyone have a list of these ids that we're going to want to set
like this? Oren? I'm a bit worried that we might want to set things
other than ids with an interface like this and that it won't transition
well.

-- Dave

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

---

Subject: Re: [PATCH 0/4] - v2 - Object creation with a specified id
Posted by Nadia Derbey on Mon, 21 Apr 2008 11:32:41 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Fri, 2008-04-18 at 07:44 +0200, Nadia.Derbey@bull.net wrote:
>
>>The syntax is one of:
>>   . echo "LONG XX" > /proc/self/task/<my_tid>/next_id
>>   next object to be created will have an id set to XX
>>   . echo "LONG<n> X0 ... X<n-1>" > /proc/self/task/<my_tid>/next_id
>>   next object to be created will have its ids set to XX0, ... X<n-1>
>>   This is particularly useful for processes that may have several ids if
>>   they belong to nested namespaces.
>
>
> I still think, in its current form, this is pretty dangerous.
>
> It might be nice to have a central place to set ids, but there's no real
> safety here for when we get a mishmash of 50 of these things. We have a
> queue and we fix the order in which things are inserted, but how do we
> control the order in which the kernel extracts them?

I thought that we were the ones who are going to be controlling the
restart phase?

I think sysV IPCs are particular, since they could be recreated early
during the restart phase:
we could define a callback routine that would be called by each process
being restarted and that would do:
for each ipc type:

1. extract next ipc id for my uid and current ipc type
2. write(fd_next_id, id, id_sz)
3. call the XXXget() routine for current ipc_type
4. remove that id from the checkpoint file

I think this could only be applied to msg queues and sems, since the creator pid is stored in the perm structure in the shm case. But in the shm case, we can replace in 1. "my uid" by "my tgid_vnr"

Then the processes hierarchy restore would take place, so no confusion between the ids.

> Can we 100%
> conform to that order in the future and be consistent? Won't that also
> be dictated by the way userspace behaves?
>
> If we're going to go this route, I think we need to do a few things.
> First, these need to be more "type safe". That means that sticking an
> ids in here that you mean to go to an IPC should never, ever, ever be
> able to get interpreted as a PID or as an inode number or anything else.
>
> echo "IPC_SEM_ID NN" > /proc/self/task/<my_tid>/next_id
> echo "PID MM" > /proc/self/task/<my_tid>/next_id
>
> Maybe we should include the types in there as well, but those are kinda
> implied by the kind of id you're setting. Have you thought about 32-bit
> userspace with 64-bit kernels?

No, but the ids are int in the ipc case and in the pid_t case too.

> Does that cause any issues?
>
> I also feel like we should have the kernel able to enumerate which
> things in a process *are* settable. What are the valid values for that
> first word being echoed in the file?
>
> Does anyone have a list of these ids that we're going to want to set
> like this? Oren? I'm a bit worried that we might want to set things
> other than ids with an interface like this and that it won't transition
> well.
>

Regards,
Nadia

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

Subject: Checkpoint/restart (was Re: [PATCH 0/4] - v2 - Object creation with a specified id)
Posted by Alexey Dobriyan on Tue, 22 Apr 2008 18:44:29 GMT
View Forum Message <> Reply to Message

On Fri, Apr 18, 2008 at 07:44:59AM +0200, Nadia.Derbey@bull.net wrote:
> . echo "LONG XX" > /proc/self/task/<my_tid>/next_id
> next object to be created will have an id set to XX
> . echo "LONG<n> X0 ... X<n-1>" > /proc/self/task/<my_tid>/next_id
> next object to be created will have its ids set to XX0, ... X<n-1>
> This is particularly useful for processes that may have several ids if
> they belong to nested namespaces.

Can we answer the following questions before merging this patch:

a) should mainline kernel have checkpoint/restart feature at all
b) if yes, should it be done from kernel- or userspace?

Until agreement will be "yes/from userspace" such patches don't make
sense in mainline.

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

Subject: Re: Checkpoint/restart (was Re: [PATCH 0/4] - v2 - Object creation with a specified id)
Posted by Dave Hansen on Tue, 22 Apr 2008 18:56:20 GMT
View Forum Message <> Reply to Message

On Tue, 2008-04-22 at 23:36 +0400, Alexey Dobriyan wrote:
> On Fri, Apr 18, 2008 at 07:44:59AM +0200, Nadia.Derbey@bull.net wrote:
> > . echo "LONG XX" > /proc/self/task/<my_tid>/next_id
> > next object to be created will have an id set to XX
> > . echo "LONG<n> X0 ... X<n-1>" > /proc/self/task/<my_tid>/next_id
> > next object to be created will have its ids set to XX0, ... X<n-1>
> > This is particularly useful for processes that may have several ids if
> > they belong to nested namespaces.
>
> Can we answer the following questions before merging this patch:

>
> a) should mainline kernel have checkpoint/restart feature at all
> b) if yes, should it be done from kernel- or userspace?
>
> Until agreement will be "yes/from userspace" such patches don't make
> sense in mainline.

What do you mean by "from kernel" or "from userspace"? Userspace has to
be involved somewhere, right? It at least need to open() the checkpoint
file and pass it into the kernel.

-- Dave

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

---

Subject: Re: Checkpoint/restart (was Re: [PATCH 0/4] - v2 - Object creation with a specified id)
Posted by serue on Tue, 22 Apr 2008 19:51:42 GMT
View Forum Message <> Reply to Message

Quoting Dave Hansen (dave@linux.vnet.ibm.com):
> On Tue, 2008-04-22 at 23:36 +0400, Alexey Dobriyan wrote:
> > On Fri, Apr 18, 2008 at 07:44:59AM +0200, Nadia.Derbey@bull.net wrote:
> > > . echo "LONG XX" > /proc/self/task/<my_tid>/next_id
> > > next object to be created will have an id set to XX
> > > . echo "LONG<n> X0 ... X<n-1>" > /proc/self/task/<my_tid>/next_id
> > > next object to be created will have its ids set to XX0, ... X<n-1>
> > > This is particularly useful for processes that may have several ids if
> > > they belong to nested namespaces.
> >
> > Can we answer the following questions before merging this patch:
> >
> > a) should mainline kernel have checkpoint/restart feature at all
> > b) if yes, should it be done from kernel- or userspace?
> >
> > Until agreement will be "yes/from userspace" such patches don't make
> > sense in mainline.
>
> What do you mean by "from kernel" or "from userspace"? Userspace has to
> be involved somewhere, right? It at least need to open() the checkpoint
> file and pass it into the kernel.

Well let's start by answering a) :

I vote yes :-)

As for b), good question. Except as Dave points out I think it's pretty unlikely that we'll end up with "sys_checkpoint(checkpoint_dirnam)" and "sys_restart(checkpointed_dirnam)". Anything more user-space driven will likely require userspace to set ids on resources, right?

But yes, we have to completely answer b) before this patch goes in, no doubt about that.

Alexey, are you advocating for a completely kernel-driven approach, or just making sure we come to a decision?

thanks,
-serge

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

---

Subject: Re: Checkpoint/restart (was Re: [PATCH 0/4] - v2 - Object creation with a specified id)
Posted by Alexey Dobriyan on Tue, 22 Apr 2008 20:09:21 GMT
View Forum Message <> Reply to Message

On Tue, Apr 22, 2008 at 11:56:20AM -0700, Dave Hansen wrote:
> On Tue, 2008-04-22 at 23:36 +0400, Alexey Dobriyan wrote:
> > On Fri, Apr 18, 2008 at 07:44:59AM +0200, Nadia.Derbey@bull.net wrote:
> > > . echo "LONG XX" > /proc/self/task/<my_tid>/next_id
> > > next object to be created will have an id set to XX
> > > . echo "LONG<n> X0 ... X<n-1>" > /proc/self/task/<my_tid>/next_id
> > > next object to be created will have its ids set to XX0, ... X<n-1>
> > > This is particularly useful for processes that may have several ids if
> > > they belong to nested namespaces.
> >
> > Can we answer the following questions before merging this patch:
> >
> > a) should mainline kernel have checkpoint/restart feature at all
> > b) if yes, should it be done from kernel- or userspace?
> >
> > Until agreement will be "yes/from userspace" such patches don't make
> > sense in mainline.
>
> What do you mean by "from kernel" or "from userspace"?

By "from userspace" I mean proposed interfaces and similar: usespace by special system calls puts some state of future object and then does

normal system call which creates aforementioned object.

This can be used for PIDs, OK.

This can be used for SystemV shmem ids. But SystemV shmem also has
(let's choose) uid/gid, atimes and actual content. How would restoration
look like?

How to restore struct task_struct::did_exec ? Do execve(2)?

A was ptracing B at checkpoint moment...

Netdevices: stats, name, all sorts of flags, hw addresses, MTU

iptables rules.

These next ids are suitable, well, only for ids which is very, very small
part of kernel state needed to restore group of processes.

> Userspace has to be involved somewhere, right? It at least need to open()
> the checkpoint file and pass it into the kernel.

Sure, but opening dumpfile is not an interesting part.

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

---

Subject: Re: Checkpoint/restart (was Re: [PATCH 0/4] - v2 - Object creation with a specified id)
Posted by Dave Hansen on Tue, 22 Apr 2008 22:56:07 GMT
View Forum Message <> Reply to Message

On Wed, 2008-04-23 at 01:01 +0400, Alexey Dobriyan wrote:
> On Tue, Apr 22, 2008 at 11:56:20AM -0700, Dave Hansen wrote:
> > On Tue, 2008-04-22 at 23:36 +0400, Alexey Dobriyan wrote:> >
> > > a) should mainline kernel have checkpoint/restart feature at all
> > > b) if yes, should it be done from kernel- or userspace?
> > >
> > > Until agreement will be "yes/from userspace" such patches don't make
> > > sense in mainline.
> >
> > What do you mean by "from kernel" or "from userspace"?
>
> By "from userspace" I mean proposed interfaces and similar: usespace by
> special system calls puts some state of future object and then does

> normal system call which creates aforementioned object.
>
> This can be used for PIDs, OK.
>
> This can be used for SystemV shmem ids. But SystemV shmem also has
> (let's choose) uid/gid, atimes and actual content. How would restoration
> look like?
>
> How to restore struct task_struct::did_exec ? Do execve(2)?
>
> A was ptracing B at checkpoint moment...
>
> Netdevices: stats, name, all sorts of flags, hw addresses, MTU
>
> iptables rules.

Don't we already have interfaces to dump these out and restore them?

My argument is this: If we have interfaces that exist (like setting up iptables rules) we shouldn't make a second interface *just* for checkpoint/restart. We have a hard enough time getting *one* interface right for things, I can't imagine getting two right, and *keeping* them right.

If the current interface is insufficient, we should first expand it in such a way that it can be used for checkpoint. That certainly won't work in all cases. fork(), for instance, doesn't take any arguments and is going to be awfully hard to expand. :)

I'd love to hear some of your insights about how things like the current iptables interfaces are insufficient for checkpoint/restart.

> These next ids are suitable, well, only for ids which is very, very small
> part of kernel state needed to restore group of processes.

I couldn't agree more. This id setting mechanism would only be useful for a small subset of the things we need during a restart.

-- Dave

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

---

Subject: Re: Checkpoint/restart (was Re: [PATCH 0/4] - v2 - Object creation with a

specified id)
Posted by Kirill Korotaev on Wed, 23 Apr 2008 06:40:21 GMT
View Forum Message <> Reply to Message

> If the current interface is insufficient, we should first expand it in
> such a way that it can be used for checkpoint. That certainly won't
> work in all cases. fork(), for instance, doesn't take any arguments and
> is going to be awfully hard to expand. :)
>
> I'd love to hear some of your insights about how things like the current
> iptables interfaces are insufficient for checkpoint/restart.

iptables is a bad example. Luckily for checkpointing - it always had an interface
"load full state", "dump full state".

But even iptables are not working in current form for checkpointing - they can't
save/restore state of conntracks. Do you think netdev@/netfilter@ guys will be happy
to have APIs allowing to set conntracks and have all the pain related
to API stability - cause conntrack state changed a couple of times during last 2 years.

Consider more intimate kernel states like:
a. task statistics
b. task start time
c. load average
d. skb state and it's data.
e. mount tree.

If you think over, e.g. (b) is a bad thing. It was used to be accounted in jiffies, then in timespec.
(a) is another example of dataset which we can't predict. task statistics change over a time.
Why bother with such intimate data in user-space at all?
Why the hell user-space should know about it and be ABLE to modify it?

Why do we need to export ability to set IDs for some objects which none of the
operating systems do and then to have a burden to support it for application compatibility
the rest of our lifes? Do you really believe none of the applications except for checkpointing
will be using it?

My personal vision is that:
1. user space must initialize checkpointing/restore state via some system call,
   supply file descriptor from where data can be read/written to.
2. must call the syscall asking kernel to restore/save different subsytems one by one.
3. finalize cpt/restore state via the syscall
But user-space MUST NOT bother about data content. At least not about the data supplied by the
kernel.
It can add additional sections if needed, e.g. about iptables state.

Having all this functionality in a signle syscall we specifically CLAIM a black box,
and that no one can use this interfaces for something different from checkpoint/restore.

So I think we have to know what other maintainers think before we can go.

>> These next ids are suitable, well, only for ids which is very, very small
>> part of kernel state needed to restore group of processes.
>
> I couldn't agree more. This id setting mechanism would only be useful
> for a small subset of the things we need during a restart.
>
> -- Dave
>
> _____
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containers
>

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

---

Subject: Re: [PATCH 0/4] - v2 - Object creation with a specified id
Posted by Pavel Machek on Wed, 23 Apr 2008 14:23:51 GMT
View Forum Message <> Reply to Message

Hi!

> When restarting a process that has been previously checkpointed, that process
> should keep on using some of its ids (such as its process id, or sysV ipc ids).
>
> This patch provides a feature that can help ensuring this saved state reuse:
> it makes it possible to create an object with a pre-defined id.
>
> A first implementation had been proposed 2 months ago. It consisted in
> changing an object's id after it had been created.
>
> Here is a second implementation based on Oren Ladaan's idea: Oren's suggestion
> was to force an object's id during its creation, rather than 1. create it,
> 2. change its id.
>
> A new file is created in procfs: /proc/self/task/<my_tid>/next_id.
> This makes it possible to avoid races between several threads belonging to
> the same process.

Ugly...

> When this file is filled with an id value, a structure pointed to by the

> calling task_struct is filled with that id.
>
> Then, when an object supporting this feature is created, the id present in
> that new structure is used, instead of the default one.
>
> The syntax is one of:
> . echo "LONG XX" > /proc/self/task/<my_tid>/next_id
> next object to be created will have an id set to XX
> . echo "LONG<n> X0 ... X<n-1>" > /proc/self/task/<my_tid>/next_id
> next object to be created will have its ids set to XX0, ... X<n-1>
> This is particularly useful for processes that may have several ids if
> they belong to nested namespaces.

So ugly it is not even funny.

Create fork_with_pid(int pid) and corresponding ipc primitives, if
you have to...
--
(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: Checkpoint/restart (was Re: [PATCH 0/4] - v2 - Object creation with a specified id)
Posted by Dave Hansen on Wed, 23 Apr 2008 15:33:18 GMT
View Forum Message <> Reply to Message

On Wed, 2008-04-23 at 10:40 +0400, Kirill Korotaev wrote:
> Having all this functionality in a signle syscall we specifically CLAIM a black box,
> and that no one can use this interfaces for something different from checkpoint/restore.
>
> So I think we have to know what other maintainers think before we can go.

I completely agree.

Have you asked any particular maintainers what they think?

-- Dave

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

---

Subject: Re: Checkpoint/restart (was Re: [PATCH 0/4] - v2 - Object creation with a specified id)
Posted by Oren Laadan on Thu, 24 Apr 2008 01:19:51 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:
>> If the current interface is insufficient, we should first expand it in
>> such a way that it can be used for checkpoint. That certainly won't
>> work in all cases. fork(), for instance, doesn't take any arguments and
>> is going to be awfully hard to expand. :)
>>
>> I'd love to hear some of your insights about how things like the current
>> iptables interfaces are insufficient for checkpoint/restart.
>
> iptables is a bad example. Luckily for checkpointing - it always had an interface
> "load full state", "dump full state".
>
> But even iptables are not working in current form for checkpointing - they can't
> save/restore state of conntracks. Do you think netdev@/netfilter@ guys will be happy
> to have APIs allowing to set conntracks and have all the pain related
> to API stability - cause conntrack state changed a couple of times during last 2 years.
>
> Consider more intimate kernel states like:
> a. task statistics
> b. task start time
> c. load average
> d. skb state and it's data.
> e. mount tree.
>
> If you think over, e.g. (b) is a bad thing. It was used to be accounted in jiffies, then in timespec.
> (a) is another example of dataset which we can't predict. task statistics change over a time.
> Why bother with such intimate data in user-space at all?
> Why the hell user-space should know about it and be ABLE to modify it?

Agreed.

> Why do we need to export ability to set IDs for some objects which none of the
> operating systems do and then to have a burden to support it for application compatibility
> the rest of our lifes? Do you really believe none of the applications except for checkpointing
> will be using it?
>
> My personal vision is that:
> 1. user space must initialize checkpointing/restore state via some system call,
>   supply file descriptor from where data can be read/written to.
> 2. must call the syscall asking kernel to restore/save different subsytems one by one.
> 3. finalize cpt/restore state via the syscall
> But user-space MUST NOT bother about data content. At least not about the data supplied by the kernel.
> It can add additional sections if needed, e.g. about iptables state.

I mostly agree with the vision that checkpoint/restart is probably best
implemented as a black box:

* First, much of the work required to restore the state of a process
as well as the state of its resources, requires kernel interfaces that
are lower than the ones available to user-space. Working in user-space
will require that we design new complex interfaces for this purpose only.

* Second, much of the state that needs to be saved was not, is not, and
should probably never be exported to user-space (e.g. interval socket
buffers, t->did_exec and many more). It is only accessible to kernel
code, so an in-kernel module (for checkpoint/restart) makes sense. It is
that sort of internals that may (and will) change as the kernel evolves
- precisely because it is not visible to user-space and not bound to it.

That said, we should still attempt to reuse existing kernel interfaces
and mechanisms as much as possible to save - and restore - the state of
processes, and prefer that over handcrafting special code. This is
especially true for restart: in checkpoint one has to _capture_ the
state by probing it in a passive manner; in contrast, in restart one
has to actively _construct_ new resources and ensure that their state
matches the saved state.

For instance, it is possible to create the process tree in a container
during restart from user-space reusing clone() (I'd argue that it's
even better to do so from user-space). Likewise, it is possible to redo
an open file by opening the file then using dup2() syscall to adjust
the target fd if necessary. The two differ in that the latter allows
to adjust the (so called) resources identifier to the desired value
(because it is privately visible), while the former does not - it gives
a globally visible identifier. And this is precisely why this thread
had started: how to determine the resource identifier when requesting
to allocate a resource (in this example, the pid).

>
> Having all this functionality in a signle syscall we specifically CLAIM a black box,
> and that no one can use this interfaces for something different from checkpoint/restore.

True, except for what can be done (and is easier to actually that way)
in user space; the most obvious example being clone() and setsid() -
which are a pain to adjust after the fact. In particular, everything
that is private in the kernel now (un-exported) should remain that way,
unless there is an (other) compelling reason to expose it.
(see http://www.ncl.cs.columbia.edu/publications/usenix2007_fordist.pdf)

Regardless of this black-box approach, we need a method to tell the
kernel code that we reuse, that we want a certain resource to have a

certain value. Whether or not this method is exported to user space
depends on whether or not that particular resource is constructed from
user space or not.

For example, Zap uses a special per-task (task_struct) field that if
set designates the next resource identifier expected to be used. It
is set before restore of pid's (clone), IPC (all sorts) and pseudo
terminals. Of these, only clone() is called from user-space, so the
interface allows only that one to be set from user-space.

Oren.

>
> So I think we have to know what other maintainers think before we can go.
>
>
>>> These next ids are suitable, well, only for ids which is very, very small
>>> part of kernel state needed to restore group of processes.
>> I couldn't agree more. This id setting mechanism would only be useful
>> for a small subset of the things we need during a restart.
>>
>> -- Dave
>>
>> _____
>> Containers mailing list
>> Containers@lists.linux-foundation.org
>> https://lists.linux-foundation.org/mailman/listinfo/containers
>>
> _____
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containers

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

---

Subject: Re: Checkpoint/restart (was Re: [PATCH 0/4] - v2 - Object creation with a specified id)
Posted by Kirill Korotaev on Thu, 24 Apr 2008 07:00:18 GMT
View Forum Message <> Reply to Message

Yeah... we talked with Andrew yesterday. He mostly agrees with a black box.
He also said an interesting idea, that if you need compatibility
between the kernels (like we for example, support for migration from 2.6.9 to 2.6.18 in OpenVZ)
you can do image conversion in user-space... Though I think we will prefer simply to have
a compatibility patch for specific kernel versions in our kernel tree (not in mainstream).

Definitely, other ideas/opinions are welcome.

Kirill

Dave Hansen wrote:
> On Wed, 2008-04-23 at 10:40 +0400, Kirill Korotaev wrote:
>> Having all this functionality in a signle syscall we specifically CLAIM a black box,
>> and that no one can use this interfaces for something different from checkpoint/restore.
>>
>> So I think we have to know what other maintainers think before we can go.
>
> I completely agree.
>
> Have you asked any particular maintainers what they think?
>
> -- Dave
>
>

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

---

Subject: Re: Checkpoint/restart (was Re: [PATCH 0/4] - v2 - Object creation with a specified id)
Posted by Dave Hansen on Thu, 24 Apr 2008 18:30:09 GMT
View Forum Message <> Reply to Message

On Thu, 2008-04-24 at 11:00 +0400, Kirill Korotaev wrote:
> Yeah... we talked with Andrew yesterday. He mostly agrees with a black
> box.
> He also said an interesting idea, that if you need compatibility
> between the kernels (like we for example, support for migration from
> 2.6.9 to 2.6.18 in OpenVZ)
> you can do image conversion in user-space...

That sounds compelling to me. I'm definitely willing to explore it.

Any thoughts on what we should start with? What can we get merged, most easily?

-- Dave

_____
Containers mailing list

Subject: Re: Checkpoint/restart (was Re: [PATCH 0/4] - v2 - Object creation with a specified id)
Posted by Oren Laadan on Thu, 24 Apr 2008 23:13:01 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Thu, 2008-04-24 at 11:00 +0400, Kirill Korotaev wrote:
>> Yeah... we talked with Andrew yesterday. He mostly agrees with a black
>> box.
>> He also said an interesting idea, that if you need compatibility
>> between the kernels (like we for example, support for migration from
>> 2.6.9 to 2.6.18 in OpenVZ)
>> you can do image conversion in user-space...
>
> That sounds compelling to me. I'm definitely willing to explore it.

Filtering/converting in user-space has long been part of zap: used to
be able to migrate between both minor and major versions of the kernel
(always upwards, of course).

In fact, I already proposed it as part of the original thread that led
eventually to the patch in question:
https://lists.linux-foundation.org/pipermail/containers/2008-February/009849.html

The need to be able to specify a desired ID (from user-space and/or from
in-kernel) remains.

Oren.

> Any thoughts on what we should start with? What can we get merged, most
> easily?
>
> -- Dave
>
> _____
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containers

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

Subject: Re: Checkpoint/restart (was Re: [PATCH 0/4] - v2 - Object creation with a specified id)
Posted by ebiederm on Thu, 10 Jul 2008 01:58:55 GMT
View Forum Message <> Reply to Message

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

>> Consider more intimate kernel states like:
>> a. task statistics
>> b. task start time
>> c. load average
>> d. skb state and it's data.
>> e. mount tree.
>>
>> If you think over, e.g. (b) is a bad thing. It was used to be accounted in
> jiffies, then in timespec.
>> (a) is another example of dataset which we can't predict. task statistics
> change over a time.
>> Why bother with such intimate data in user-space at all?
>> Why the hell user-space should know about it and be ABLE to modify it?
>
> Agreed.

Almost agreed. The reason we care is that the data is visible to user space in some form. So we need to save it, but hopefully not in it's internal kernel representation. If we don't care at all we should not save it.

>> My personal vision is that:
>> 1. user space must initialize checkpointing/restore state via some system
> call,
>> supply file descriptor from where data can be read/written to.
>> 2. must call the syscall asking kernel to restore/save different subsytems one
> by one.
>> 3. finalize cpt/restore state via the syscall
>> But user-space MUST NOT bother about data content. At least not about the data
> supplied by the kernel.
>> It can add additional sections if needed, e.g. about iptables state.
>
> I mostly agree with the vision that checkpoint/restart is probably best
> implemented as a black box:

I would claim an atomic unit. Roughly like a coredump is today. We know what a core dump does and we don't care how it does it.

> * First, much of the work required to restore the state of a process
> as well as the state of its resources, requires kernel interfaces that
> are lower than the ones available to user-space. Working in user-space
> will require that we design new complex interfaces for this purpose only.

Yes.

> * Second, much of the state that needs to be saved was not, is not, and
> should probably never be exported to user-space (e.g. interval socket
> buffers, t->did_exec and many more). It is only accessible to kernel
> code, so an in-kernel module (for checkpoint/restart) makes sense. It is
> that sort of internals that may (and will) change as the kernel evolves
> - precisely because it is not visible to user-space and not bound to it.

No. If the state can be inferred from user space it is visible to user
space. However there is state visible to user space like did_exec
that is not directly manipulatable by user space.

In the worst case today we can restore a checkpoint by replaying all of
the user space actions that took us to get there. That is a tedious
and slow approach.

> That said, we should still attempt to reuse existing kernel interfaces
> and mechanisms as much as possible to save - and restore - the state of
> processes, and prefer that over handcrafting special code. This is
> especially true for restart: in checkpoint one has to _capture_ the
> state by probing it in a passive manner; in contrast, in restart one
> has to actively _construct_ new resources and ensure that their state
> matches the saved state.

> For instance, it is possible to create the process tree in a container
> during restart from user-space reusing clone() (I'd argue that it's
> even better to do so from user-space). Likewise, it is possible to redo
> an open file by opening the file then using dup2() syscall to adjust
> the target fd if necessary. The two differ in that the latter allows
> to adjust the (so called) resources identifier to the desired value
> (because it is privately visible), while the former does not - it gives
> a globally visible identifier. And this is precisely why this thread
> had started: how to determine the resource identifier when requesting
> to allocate a resource (in this example, the pid).

The limiting factor to me appears to be live migration.

As I understand the concept. Live migration is where you take you
first take a snapshot of a running system, and restore that snapshot
on another system and don't start it.

Then you repeat with an incremental snapshot and you do an incremental
restore.

Until ideally the changes are small and you can afford to have the
system paused for the amount of time it takes to transfer and restore
the last incremental snapshot.

I expect a design that allows for multiple cpus to work on the
checkpoint/restore paths and that allows for incremental and
thus live migration are going to be the limiting factors in a good
interface design.

Pushing the work into the kernel with an atomic syscall style approach
so that the normal people maintaining the kernel can have visibility
of the checkpoint/restore code paths and can do some of the work
seems healthy.

>> Having all this functionality in a signle syscall we specifically CLAIM a
> black box,
>> and that no one can use this interfaces for something different from
> checkpoint/restore.
>
> True, except for what can be done (and is easier to actually that way)
> in user space; the most obvious example being clone() and setsid() -
> which are a pain to adjust after the fact. In particular, everything
> that is private in the kernel now (un-exported) should remain that way,
> unless there is an (other) compelling reason to expose it.
> (see http://www.ncl.cs.columbia.edu/publications/usenix2007_fordist.pdf)

Yes. A very good example of something that is user visible but should
not be user settable (except during restore) is the start time for a
monotonic clock. If you allow someone to set it arbitrarily it is no
longer a monotonic clock and it looses all value.

So here is one suggestion:
sys_processtree_isolate(container_init_pid);
sys_checkpoint(old_dir_fd, new_dir_fd, container_init_pid);
sys_processtree_unisolate(container_init_pid);

container_init_pid = sys_restore(dir_fd, old_container_init_pid);
sys_processtree_unisolate(container_init_pid);

Then save the different components in different files. And in the non-trivial cases
have tagged data in the files. The tags allow us to have different data types
easily.

For data that already have or should have an interface for
persistence. Filesystems being the primary example we don't handle
this way. Instead we use the already existing techniques to
backup/snapshot the data and then to restore the data.

Isolation should be used instead of freezing if we can, because
isolation is a much cheaper concept, and it allows for multi-machine

synchronized checkpoints. On the big scale if I unplug a machines network cable (isolating it) then I don't care when between the time I isolate the machine and the time I plug the cable back in. It all looks the same to the outside world. But just the comunications of the machine were frozen. Likewise for processes. SIGSTOP is good but I suspect it may be excessive, so denies us optimization opportunities (at least in the interface).

So in summary we want an interface that allows for.
- Existing persistent kernel state to using the preexisting mechanisms.
- The persistence code to be looked after by the people working on the kernel subsystem.
- Migration between kernel versions, (possibly with user space intervention).
- Live migration
- Minimal intervention of the processes being checkpointed.
- Denying persistence to applications that need it.
- Checkpoints coordinated between multiple containers or real machines.
- Documented in terms of isolation rather then freezing, as it is the isolation that is the important property.
- Restricted access to user space visible but immutable state.
- Ideally sufficiently general that debuggers can take advantage of the checkpoints.

Eric

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

---

Subject: Re: Checkpoint/restart (was Re: [PATCH 0/4] - v2 - Object creation with a specified id)

Posted by Dave Hansen on Thu, 10 Jul 2008 17:12:29 GMT

View Forum Message <> Reply to Message

On Wed, 2008-07-09 at 18:58 -0700, Eric W. Biederman wrote:
> In the worst case today we can restore a checkpoint by replaying all of
> the user space actions that took us to get there. That is a tedious
> and slow approach.

Yes, tedious and slow, *and* minimally invasive in the kernel. Once we have a tedious and slow process, we'll have some really good points when we try to push the next set of patches to make it less slow and tedious. We'll be able to describe an _actual_ set of problems to our fellow kernel hackers.

So, the checkpoint-as-a-corefile idea sounds good to me, but it

definitely leaves a lot of questions about exactly how we'll need to do
the restore.

-- Dave

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

---

Subject: Re: Checkpoint/restart (was Re: [PATCH 0/4] - v2 - Object creation with a specified id)
Posted by serue on Thu, 10 Jul 2008 17:32:46 GMT
View Forum Message <> Reply to Message

Quoting Dave Hansen (dave@linux.vnet.ibm.com):
> On Wed, 2008-07-09 at 18:58 -0700, Eric W. Biederman wrote:
> > In the worst case today we can restore a checkpoint by replaying all of
> > the user space actions that took us to get there. That is a tedious
> > and slow approach.
>
> Yes, tedious and slow, *and* minimally invasive in the kernel. Once we
> have a tedious and slow process, we'll have some really good points when
> we try to push the next set of patches to make it less slow and tedious.
> We'll be able to describe an _actual_ set of problems to our fellow
> kernel hackers.
>
> So, the checkpoint-as-a-corefile idea sounds good to me, but it
> definitely leaves a lot of questions about exactly how we'll need to do
> the restore.

Talking with Dave over irc, I kind of liked the idea of creating a new
fs/binfmt_cr.c that executes a checkpoint-as-a-coredump file.

One thing I do not like about the checkpoint-as-coredump is that it begs
us to dump all memory out into the file. Our plan/hope was to save
ourselves from writing out most memory by:

1. associating a separate swapfile with each container
2. doing a swapfile snapshot at each checkpoint
3. dumping the pte entries (/proc/self/)

If we do checkpoint-as-a-coredump, then we need userspace to coordinate
a kernel-generated coredump with a user-generated (?) swapfile snapshot.
But I guess we figure that out later.

-serge

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

Subject: Re: Checkpoint/restart (was Re: [PATCH 0/4] - v2 - Object creation with a specified id)
Posted by ebiederm on Thu, 10 Jul 2008 18:55:34 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serue@us.ibm.com> writes:

>> So, the checkpoint-as-a-corefile idea sounds good to me, but it
>> definitely leaves a lot of questions about exactly how we'll need to do
>> the restore.
>
> Talking with Dave over irc, I kind of liked the idea of creating a new
> fs/binfmt_cr.c that executes a checkpoint-as-a-coredump file.
>
> One thing I do not like about the checkpoint-as-coredump is that it begs
> us to dump all memory out into the file. Our plan/hope was to save
> ourselves from writing out most memory by:
>
> 1. associating a separate swapfile with each container
> 2. doing a swapfile snapshot at each checkpoint
> 3. dumping the pte entries (/proc/self/)
>
> If we do checkpoint-as-a-coredump, then we need userspace to coordinate
> a kernel-generated coredump with a user-generated (?) swapfile snapshot.
> But I guess we figure that out later.

Well it is a matter of which VMAs you dump. For things that are file backed
you need to dump them.

I don't know that even a binfmt for per process level checkpoints is sufficient
but I do know having something of that granularity looks much easier. Otherwise
it takes a bazillian little syscalls to do things no one else is interested in doing.

Eric

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

Subject: Re: Checkpoint/restart (was Re: [PATCH 0/4] - v2 - Object creation with a

specified id)
Posted by Dave Hansen on Thu, 10 Jul 2008 19:06:34 GMT
View Forum Message <> Reply to Message

On Thu, 2008-07-10 at 11:55 -0700, Eric W. Biederman wrote:
>
> > If we do checkpoint-as-a-coredump, then we need userspace to coordinate
> > a kernel-generated coredump with a user-generated (?) swapfile snapshot.
> > But I guess we figure that out later.
>
> Well it is a matter of which VMAs you dump. For things that are file backed
> you need to dump them.

Are we talking about the VMA itself, or the memory backing the VMA?

-- Dave

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

Subject: Re: Checkpoint/restart (was Re: [PATCH 0/4] - v2 - Object creation with a specified id)
Posted by ebiederm on Thu, 10 Jul 2008 19:21:43 GMT
View Forum Message <> Reply to Message

Dave Hansen <dave@linux.vnet.ibm.com> writes:

> On Thu, 2008-07-10 at 11:55 -0700, Eric W. Biederman wrote:
>>
>> > If we do checkpoint-as-a-coredump, then we need userspace to coordinate
>> > a kernel-generated coredump with a user-generated (?) swapfile snapshot.
>> > But I guess we figure that out later.
>>
>> Well it is a matter of which VMAs you dump. For things that are file backed
>> you need to dump them.

For things that are file backed you DO NOT need to dump them. (sorry typo).

> Are we talking about the VMA itself, or the memory backing the VMA?

The memory backing the VMA. We need to store the page protections
that the memory was mapped with as well now that you point it out. A
VMA is not user space visible, which is why we can arbitrarily split
and merge VMAs.

Eric

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

Subject: Re: Checkpoint/restart (was Re: [PATCH 0/4] - v2 - Object creation with a specified id)
Posted by Dave Hansen on Thu, 10 Jul 2008 19:47:32 GMT
View Forum Message <> Reply to Message

On Thu, 2008-07-10 at 12:21 -0700, Eric W. Biederman wrote:
> > Are we talking about the VMA itself, or the memory backing the VMA?
>
> The memory backing the VMA. We need to store the page protections
> that the memory was mapped with as well now that you point it out. A
> VMA is not user space visible, which is why we can arbitrarily split
> and merge VMAs.

It is visible with /proc/$pid/{maps,smaps,numamaps?}. That's the only efficient way I know of from userspace to figure out where userspace's memory is and if it *is* file backed, and what the permissions are.

We also can't truly split them arbitrarily because the memory cost of the VMAs themselves might become too heavy (the remap_file_pages() problem).

It gets trickier when things are also private mappings in addition to being in a file-backed VMA. We *do* need to checkpoint those, but only the pages to which there was a write.

There's also the problem of restoring things read-only, but VM_MAYWRITE. If there's a high level of (non-COW'd) sharing of these anonymous areas, we may not be able to even restore the set of processes unless we can replicate the sharing. We might run out of memory if the sharing isn't replicated.

Memory is fun! :)

-- Dave

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

Subject: Re: Checkpoint/restart (was Re: [PATCH 0/4] - v2 - Object creation with a specified id)
Posted by Alexey Dobriyan on Fri, 11 Jul 2008 00:32:58 GMT
View Forum Message <> Reply to Message

On Thu, Jul 10, 2008 at 12:47:32PM -0700, Dave Hansen wrote:
> On Thu, 2008-07-10 at 12:21 -0700, Eric W. Biederman wrote:
> > > Are we talking about the VMA itself, or the memory backing the VMA?
> >
> > The memory backing the VMA. We need to store the page protections
> > that the memory was mapped with as well now that you point it out. A
> > VMA is not user space visible, which is why we can arbitrarily split
> > and merge VMAs.
>
> It is visible with /proc/$pid/{maps,smaps,numamaps?}. That's the only
> efficient way I know of from userspace to figure out where userspace's
> memory is and if it *is* file backed, and what the permissions are.

None of those files exist if PROC_FS is off.

> We also can't truly split them arbitrarily because the memory cost of
> the VMAs themselves might become too heavy (the remap_file_pages()
> problem).
>
> It gets trickier when things are also private mappings in addition to
> being in a file-backed VMA. We *do* need to checkpoint those, but only
> the pages to which there was a write.
>
> There's also the problem of restoring things read-only, but VM_MAYWRITE.
> If there's a high level of (non-COW'd) sharing of these anonymous areas,
> we may not be able to even restore the set of processes unless we can
> replicate the sharing. We might run out of memory if the sharing isn't
> replicated.

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

Subject: Re: Checkpoint/restart (was Re: [PATCH 0/4] - v2 - Object creation with a specified id)
Posted by Oren Laadan on Thu, 17 Jul 2008 23:09:53 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Oren Laadan <orenl@cs.columbia.edu> writes:
>

>>> Consider more intimate kernel states like:
>>> a. task statistics
>>> b. task start time
>>> c. load average
>>> d. skb state and it's data.
>>> e. mount tree.
>>>
>>> If you think over, e.g. (b) is a bad thing. It was used to be accounted in
>> jiffies, then in timespec.
>>> (a) is another example of dataset which we can't predict. task statistics
>> change over a time.
>>> Why bother with such intimate data in user-space at all?
>>> Why the hell user-space should know about it and be ABLE to modify it?
>> Agreed.
>
> Almost agreed. The reason we care is that the data is visible to user
> space in some form. So we need to save it, but hopefully not in it's internal
> kernel representation. If we don't care at all we should not save it.

yes, we probably prefer an intermediate representation to some extent, mainly
because we (a) don't want garbage data that we don't use and (b) to make the
task of converting from old to new kernel version easier.

But only for these two reasons. I see zero value in avoiding representation
used internally by the kernel for the sake of avoiding it. In fact, if it's
that much easier to stick to that representation in a specific case -- so be
it. There shall be userland utilities that will be able to inspect the contents
(like those that can inspect the contents of internal file system data).

>>> My personal vision is that:
>>> 1. user space must initialize checkpointing/restore state via some system
>> call,
>>>   supply file descriptor from where data can be read/written to.
>>> 2. must call the syscall asking kernel to restore/save different subsytems one
>> by one.
>>> 3. finalize cpt/restore state via the syscall
>>> But user-space MUST NOT bother about data content. At least not about the data
>> supplied by the kernel.
>>> It can add additional sections if needed, e.g. about iptables state.
>> I mostly agree with the vision that checkpoint/restart is probably best
>> implemented as a black box:
>
> I would claim an atomic unit. Roughly like a coredump is today. We know
> what a core dump does and we don't care how it does it.
>
>> * First, much of the work required to restore the state of a process
>> as well as the state of its resources, requires kernel interfaces that
>> are lower than the ones available to user-space. Working in user-space

>> will require that we design new complex interfaces for this purpose only.
>
> Yes.
>
>> * Second, much of the state that needs to be saved was not, is not, and
>> should probably never be exported to user-space (e.g. interval socket
>> buffers, t->did_exec and many more). It is only accessible to kernel
>> code, so an in-kernel module (for checkpoint/restart) makes sense. It is
>> that sort of internals that may (and will) change as the kernel evolves
>> - precisely because it is not visible to user-space and not bound to it.
>
> No. If the state can be inferred from user space it is visible to user
> space. However there is state visible to user space like did_exec
> that is not directly manipulatable by user space.

Not all state can be inferred; and the point is that I don't want to need
to chase the kernel devs and add such interfaces every time some state is
added.

>
> In the worst case today we can restore a checkpoint by replaying all of
> the user space actions that took us to get there. That is a tedious
> and slow approach.

ugh ... not only tedious - but not entirely correct: deterministic replay is
a very complicated problem (but thrilling) ! and you'll need to log/record
everything during the execution :(

>
>> That said, we should still attempt to reuse existing kernel interfaces
>> and mechanisms as much as possible to save - and restore - the state of
>> processes, and prefer that over handcrafting special code. This is
>> especially true for restart: in checkpoint one has to _capture_ the
>> state by probing it in a passive manner; in contrast, in restart one
>> has to actively _construct_ new resources and ensure that their state
>> matches the saved state.
>
>> For instance, it is possible to create the process tree in a container
>> during restart from user-space reusing clone() (I'd argue that it's
>> even better to do so from user-space). Likewise, it is possible to redo
>> an open file by opening the file then using dup2() syscall to adjust
>> the target fd if necessary. The two differ in that the latter allows
>> to adjust the (so called) resources identifier to the desired value
>> (because it is privately visible), while the former does not - it gives
>> a globally visible identifier. And this is precisely why this thread
>> had started: how to determine the resource identifier when requesting
>> to allocate a resource (in this example, the pid).
>

> The limiting factor to me appears to be live migration.

limiting in what sense ?

> As I understand the concept. Live migration is where you take you
> first take a snapshot of a running system, and restore that snapshot
> on another system and don't start it.
>
> Then you repeat with an incremental snapshot and you do an incremental
> restore.
>
> Until ideally the changes are small and you can afford to have the
> system paused for the amount of time it takes to transfer and restore
> the last incremental snapshot.

yes; in practice, you mostly care about memory contents that have
changed in the interim, as saving/transfering memory is by far the
most time consuming component of a checkpoint/migration.

So you must track which memory pages were modified between successive
attempts to complete the migration. As for the rest - you can either
checkpoint all the rest during the final snapshot, or save all of them
at the beginning and track changes to them too.

>
> I expect a design that allows for multiple cpus to work on the
> checkpoint/restore paths and that allows for incremental and
> thus live migration are going to be the limiting factors in a good
> interface design.

These two are important factors in my reasoning to use in kernel
implementation. Such implementation should allow concurrency (e.g.
using kernel threads), as well as the required atomicity to track
the changes between successive iterations.

> Pushing the work into the kernel with an atomic syscall style approach
> so that the normal people maintaining the kernel can have visibility
> of the checkpoint/restore code paths and can do some of the work
> seems healthy.

So now it seems that we agree on that a kernel approach is suitable.

>
>>> Having all this functionality in a signle syscall we specifically CLAIM a
>> black box,
>>> and that no one can use this interfaces for something different from
>> checkpoint/restore.
>>

>> True, except for what can be done (and is easier to actually that way)
>> in user space; the most obvious example being clone() and setsid() -
>> which are a pain to adjust after the fact. In particular, everything
>> that is private in the kernel now (un-exported) should remain that way,
>> unless there is an (other) compelling reason to expose it.
>> (see http://www.ncl.cs.columbia.edu/publications/usenix2007_fordist.pdf)
>
> Yes. A very good example of something that is user visible but should
> not be user settable (except during restore) is the start time for a
> monotonic clock. If you allow someone to set it arbitrarily it is no
> longer a monotonic clock and it looses all value.
>
> So here is one suggestion:
> sys_processtree_isolate(container_init_pid);
> sys_checkpoint(old_dir_fd, new_dir_fd, container_init_pid);
> sys_processtree_unisolate(container_init_pid);
>
> container_init_pid = sys_restore(dir_fd, old_container_init_pid);
> sys_processtree_unisolate(container_init_pid);

what do you mean by "isolate" and "unisolate" ?

>
> Then save the different components in different files. And in the non-trivial cases
> have tagged data in the files. The tags allow us to have different data types
> easily.
>

why different files ? why not a single file, streamed - beneficial
for live migration, for example, or pass the data through filters
(encryption, signature, conversion between kernel versions, etc).

> For data that already have or should have an interface for
> persistence. Filesystems being the primary example we don't handle
> this way. Instead we use the already existing techniques to
> backup/snapshot the data and then to restore the data.
>
> Isolation should be used instead of freezing if we can, because
> isolation is a much cheaper concept, and it allows for multi-machine
> synchronized checkpoints. On the big scale if I unplug a machines

if you let the processes in a container to proceed execution (even
if isolated) during a checkpoint, you will be unable to capture a
state that is consistent among all processes in that container. So
you must freeze them somehow.

> network cable (isolating it) then I don't care when between the time I
> isolate the machine and the time I plug the cable back in. It all

> looks the same to the outside world. But just the comunications of
> the machine were frozen. Likewise for processes. SIGSTOP is good

I am not aware of any simple way that can prevent any sort of interaction
between processes (by simple, I mean like unplugging a cable). For
example, how would you prevent shared memory alterations ? signals that
are raised due to async IO completion ? and many more examples.

> but I suspect it may be excessive, so denies us optimization
> opportunities (at least in the interface).
>

> So in summary we want an interface that allows for.
> - Existing persistent kernel state to using the preexisting mechanisms.
> - The persistence code to be looked after by the people working on the
>   kernel subsystem.
> - Migration between kernel versions, (possibly with user space intervention).
> - Live migration
> - Minimal intervention of the processes being checkpointed.
> - Denying persistence to applications that need it.
> - Checkpoints coordinated between multiple containers or real
>   machines.

you mean distributed checkpoint/restart ? not an easy job either, and
should not, IMHO, be done at kernel level. Perhaps you'll find this
paper interesting:
http://www.ncl.cs.columbia.edu/publications/cluster2005_fordist.pdf

> - Documented in terms of isolation rather then freezing, as it is the
>   isolation that is the important property.

can you please define "isolation" in this context ? "freezing" is very
simple: the processes cannot do anything while they are checkpointed.
(in the case of live migration, they aren't frozen except for very short
time - aka downtime).

> - Restricted access to user space visible but immutable state.

> - Ideally sufficiently general that debuggers can take advantage
>   of the checkpoints.

This last point is interesting and important, but not high priority.
It is always possible to take a checkpoint image and - in user space -
convert it to a format that is readable by debuggers. So it should not
be a guiding factor in designing the checkpoint/restart mechanism.

Oren.

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

Subject: Re: Checkpoint/restart (was Re: [PATCH 0/4] - v2 - Object creation with a specified id)
Posted by Oren Laadan on Thu, 17 Jul 2008 23:14:25 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Wed, 2008-07-09 at 18:58 -0700, Eric W. Biederman wrote:
>> In the worst case today we can restore a checkpoint by replaying all of
>> the user space actions that took us to get there. That is a tedious
>> and slow approach.
>
> Yes, tedious and slow, *and* minimally invasive in the kernel. Once we
> have a tedious and slow process, we'll have some really good points when

"Replaying all of the user space actions that took us to get there" -
this task is not even always possible without deterministic log/replay
mechanism :(

Oren.

> we try to push the next set of patches to make it less slow and tedious.
> We'll be able to describe an _actual_ set of problems to our fellow
> kernel hackers.
>
> So, the checkpoint-as-a-corefile idea sounds good to me, but it
> definitely leaves a lot of questions about exactly how we'll need to do
> the restore.
>
> -- Dave
>

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

Subject: Re: Checkpoint/restart (was Re: [PATCH 0/4] - v2 - Object creation with a specified id)
Posted by Oren Laadan on Thu, 17 Jul 2008 23:16:10 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Dave Hansen (dave@linux.vnet.ibm.com):
>> On Wed, 2008-07-09 at 18:58 -0700, Eric W. Biederman wrote:
>>> In the worst case today we can restore a checkpoint by replaying all of
>>> the user space actions that took us to get there. That is a tedious
>>> and slow approach.
>> Yes, tedious and slow, *and* minimally invasive in the kernel. Once we
>> have a tedious and slow process, we'll have some really good points when
>> we try to push the next set of patches to make it less slow and tedious.
>> We'll be able to describe an _actual_ set of problems to our fellow
>> kernel hackers.
>>
>> So, the checkpoint-as-a-corefile idea sounds good to me, but it
>> definitely leaves a lot of questions about exactly how we'll need to do
>> the restore.
>
> Talking with Dave over irc, I kind of liked the idea of creating a new
> fs/binfmt_cr.c that executes a checkpoint-as-a-coredump file.
>
> One thing I do not like about the checkpoint-as-coredump is that it begs
> us to dump all memory out into the file. Our plan/hope was to save
> ourselves from writing out most memory by:
>
> 1. associating a separate swapfile with each container
> 2. doing a swapfile snapshot at each checkpoint
> 3. dumping the pte entries (/proc/self/)
>
> If we do checkpoint-as-a-coredump, then we need userspace to coordinate
> a kernel-generated coredump with a user-generated (?) swapfile snapshot.
> But I guess we figure that out later.

I'm not sure how this approach integrates with (a) live migration (and
the iterative process of sending over memory modified since previous
iteration), and (b) incremental checkpoint (where except for the first
snapshot, additional snapshots only save what changed since the previous
one).

Oren.

>
> -serge

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

Subject: Re: Checkpoint/restart (was Re: [PATCH 0/4] - v2 - Object creation with a specified id)
Posted by Oren Laadan on Thu, 17 Jul 2008 23:19:51 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Thu, 2008-07-10 at 12:21 -0700, Eric W. Biederman wrote:
>>> Are we talking about the VMA itself, or the memory backing the VMA?
>> The memory backing the VMA. We need to store the page protections
>> that the memory was mapped with as well now that you point it out. A
>> VMA is not user space visible, which is why we can arbitrarily split
>> and merge VMAs.
>
> It is visible with /proc/$pid/{maps,smaps,numamaps?}. That's the only
> efficient way I know of from userspace to figure out where userspace's
> memory is and if it *is* file backed, and what the permissions are.
>
> We also can't truly split them arbitrarily because the memory cost of
> the VMAs themselves might become too heavy (the remap_file_pages()
> problem).
>
> It gets trickier when things are also private mappings in addition to
> being in a file-backed VMA. We *do* need to checkpoint those, but only
> the pages to which there was a write.
>
> There's also the problem of restoring things read-only, but VM_MAYWRITE.
> If there's a high level of (non-COW'd) sharing of these anonymous areas,
> we may not be able to even restore the set of processes unless we can
> replicate the sharing. We might run out of memory if the sharing isn't
> replicated.
>
> Memory is fun! :)

Heh .. and it can get much worse. By all means, the functions that analyze and save the VMAs during checkpoint and later reconstruct them during restart are the most complicated logic. The good news, however, is that it works :)

Oren.

>
> -- Dave
>

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