Subject: Roadmap for features planed for containers where and Some future features ideas.
Posted by Peter Dolding on Mon, 21 Jul 2008 11:03:47 GMT
View Forum Message <> Reply to Message

http://opensolaris.org/os/community/brandz/ I would like to see if something equal to this is on the roadmap in particular. Being able to run solaris and aix closed source binaries contained would be useful.

Other useful feature is some way to share a single process between PID containers as like a container bridge. For containers used for desktop applications not having a single X11 server interfacing with video card is a issue.

These container bridges avoid having to go threw network cards and other means to share data between containers. A user space solution.

I know this reduces secuirty but when you need a application form X distrobuton and you have Y distribution and its opengl heavy you are kinda stuffed at moment.

Final one is some form of LSM processing different. Lot of the Linux Secuirty channel talk about containers as light weight virtualisation so will never need to run a OS inside with a different LSM profile to the master OS. If containers plan to go after brandz like containers this needs to be made clear that LSM different processing will be required.

Peter Dolding

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

Subject: Re: Roadmap for features planed for containers where and Some future features ideas.
Posted by ebiederm on Mon, 21 Jul 2008 12:13:27 GMT
View Forum Message <> Reply to Message

"Peter Dolding" <oiaohm@gmail.com> writes:

> http://opensolaris.org/os/community/brandz/ I would like to see if
> something equal to this is on the roadmap in particular. Being able
> to run solaris and aix closed source binaries contained would be
> useful.

There have been projects to do this at various times on linux. Having
a namespace dedicated to a certain kind of application is no big deal.
Someone would need to care enough to test and implement it though.

> Other useful feature is some way to share a single process between PID
> containers as like a container bridge. For containers used for
> desktop applications not having a single X11 server interfacing with
> video card is a issue.

X allows network connections, and I think unix domain sockets will work.
The latter I need to check on.

The pid namespace is well defined and no a task will not be able
to change it's pid namespace while running. That is nasty.

> These container bridges avoid having to go threw network cards and
> other means to share data between containers. A user space solution.

There are lots of opportunities for user space solutions.

> I know this reduces secuirty but when you need a application form X
> distrobuton and you have Y distribution and its opengl heavy you are
> kinda stuffed at moment.
>
> Final one is some form of LSM processing different. Lot of the Linux
> Secuirty channel talk about containers as light weight virtualisation
> so will never need to run a OS inside with a different LSM profile to
> the master OS. If containers plan to go after brandz like containers
> this needs to be made clear that LSM different processing will be
> required.

We have had that discussion mostly this appears to be a measure of
matureness.

Eric

Subject: Re: Roadmap for features planed for containers where and Some future features ideas.
Posted by Peter Dolding on Mon, 21 Jul 2008 13:21:35 GMT
View Forum Message <> Reply to Message

On Mon, Jul 21, 2008 at 10:13 PM, Eric W. Biederman
<ebiederm@xmission.com> wrote:

> "Peter Dolding" <oiaohm@gmail.com> writes:
>
>> http://opensolaris.org/os/community/brandz/ I would like to see if
>> something equal to this is on the roadmap in particular. Being able
>> to run solaris and aix closed source binaries contained would be
>> useful.
>
> There have been projects to do this at various times on linux. Having
> a namespace dedicated to a certain kind of application is no big deal.
> Someone would need to care enough to test and implement it though.
>
>> Other useful feature is some way to share a single process between PID
>> containers as like a container bridge. For containers used for
>> desktop applications not having a single X11 server interfacing with
>> video card is a issue.
>
> X allows network connections, and I think unix domain sockets will work.
> The latter I need to check on.

Does to a point until you see that local X11 is using shared memory
for speed. Hardest issue is getting GLX working.

> The pid namespace is well defined and no a task will not be able
> to change it's pid namespace while running. That is nasty.
Ok if that is imposable to extremely risky.

What about a form of a proxy pid in the pid namespace proxying
application chatter between 1 name space to another. Applications
being the bridge if its not possible to do it invisible to application
could be made aware of it. So they can provide shared memory and the
like across pid namespaces. But only where they have a activated proxy
to do there bidding. This also allows applications to maintain there
own internal secuirty between namespaces.

Ie application is 1 pid number in its source container and virtual pid
numbers in the following containers. Symbolic linking at task level
yes a little warped. Yes this will annoying mean a special set of
syscalls and a special set of capabilities and restrictions. Like PID
containers starting up forbidding proxy pid's or allowing them.

If I am thinking right that avoids not be able to change it's pid.
Instead sending and receiving the messages you need in the other name
space threw a small proxy. Yes I know that will cost some
performance.

Basically want to setup a neat universal container way of handling
stuff like http://www.cs.toronto.edu/~andreslc/xen-gl/ without having
to go network and hopefully in a way that limitations don't have to

exist since messages are really only be sent threw 1 X11 server to 1
driver system. Only thing is really sending the correct messages to
the correct place. There will most likely be other services were a
single entity at times is preferred. Worst out come is if proxying
.so is required.

Peter Dolding

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

---

Subject: Re: Roadmap for features planed for containers where and Some future features ideas.
Posted by ebiederm on Tue, 22 Jul 2008 01:28:43 GMT
View Forum Message <> Reply to Message

"Peter Dolding" <oiaohm@gmail.com> writes:

> On Mon, Jul 21, 2008 at 10:13 PM, Eric W. Biederman
> <ebiederm@xmission.com> wrote:
>> "Peter Dolding" <oiaohm@gmail.com> writes:
>>
>>> http://opensolaris.org/os/community/brandz/ I would like to see if
>>> something equal to this is on the roadmap in particular. Being able
>>> to run solaris and aix closed source binaries contained would be
>>> useful.
>>
>> There have been projects to do this at various times on linux. Having
>> a namespace dedicated to a certain kind of application is no big deal.
>> Someone would need to care enough to test and implement it though.
>>
>>> Other useful feature is some way to share a single process between PID
>>> containers as like a container bridge. For containers used for
>>> desktop applications not having a single X11 server interfacing with
>>> video card is a issue.
>>
>> X allows network connections, and I think unix domain sockets will work.
>> The latter I need to check on.
>
> Does to a point until you see that local X11 is using shared memory
> for speed. Hardest issue is getting GLX working.

That is easier in general. Don't unshare the sysvipc namespace.
Or share the mount of /dev/shmem at least for the file X cares about.

>> The pid namespace is well defined and no a task will not be able

>> to change it's pid namespace while running. That is nasty.
> Ok if that is imposable to extremely risky.
>
> What about a form of a proxy pid in the pid namespace proxying
> application chatter between 1 name space to another. Applications
> being the bridge if its not possible to do it invisible to application
> could be made aware of it. So they can provide shared memory and the
> like across pid namespaces. But only where they have a activated proxy
> to do there bidding. This also allows applications to maintain there
> own internal secuirty between namespaces.
>
> Ie application is 1 pid number in its source container and virtual pid
> numbers in the following containers. Symbolic linking at task level
> yes a little warped. Yes this will annoying mean a special set of
> syscalls and a special set of capabilities and restrictions. Like PID
> containers starting up forbidding proxy pid's or allowing them.
>
> If I am thinking right that avoids not be able to change it's pid.
> Instead sending and receiving the messages you need in the other name
> space threw a small proxy. Yes I know that will cost some
> performance.

Proxy pids don't actually do anything for you, unless you want to send
signals. Because all of the namespaces are distinct. So even at the
best of it you can see the X server but it still can't use your
network sockets or ipc shm.

Better is working out the details on how to manipulate multiple
sysvipc and network namespaces from a single application. Mostly
that is supported now by the objects there is just no easy way
of dealing with it.

> Basically want to setup a neat universal container way of handling
> stuff like http://www.cs.toronto.edu/~andreslc/xen-gl/ without having
> to go network and hopefully in a way that limitations don't have to
> exist since messages are really only be sent threw 1 X11 server to 1
> driver system. Only thing is really sending the correct messages to
> the correct place. There will most likely be other services were a
> single entity at times is preferred. Worst out come is if proxying
> .so is required.

Yes. I agree that is essentially desirable. Given that I think
high end video card actually have multiple hardware contexts that
can be mapped into different user space processes there may be other
ways of handling this.

Ideally we can find a high performance solution to X that also gives
us good isolation and migration properties. Certainly something to talk

about tomorrow in the conference.

Eric

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

---

Subject: Re: Roadmap for features planed for containers where and Some future features ideas.
Posted by Oren Laadan on Tue, 22 Jul 2008 14:05:27 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> "Peter Dolding" <oiaohm@gmail.com> writes:
>
>> On Mon, Jul 21, 2008 at 10:13 PM, Eric W. Biederman
>> <ebiederm@xmission.com> wrote:
>>> "Peter Dolding" <oiaohm@gmail.com> writes:
>>>
>>>> http://opensolaris.org/os/community/brandz/ I would like to see if
>>>> something equal to this is on the roadmap in particular. Being able
>>>> to run solaris and aix closed source binaries contained would be
>>>> useful.
>>> There have been projects to do this at various times on linux. Having
>>> a namespace dedicated to a certain kind of application is no big deal.
>>> Someone would need to care enough to test and implement it though.
>>>
>>>> Other useful feature is some way to share a single process between PID
>>>> containers as like a container bridge. For containers used for
>>>> desktop applications not having a single X11 server interfacing with
>>>> video card is a issue.
>>> X allows network connections, and I think unix domain sockets will work.
>>> The latter I need to check on.
>> Does to a point until you see that local X11 is using shared memory
>> for speed. Hardest issue is getting GLX working.
>
> That is easier in general. Don't unshare the sysvipc namespace.
> Or share the mount of /dev/shmem at least for the file X cares about.
>
>>> The pid namespace is well defined and no a task will not be able
>>> to change it's pid namespace while running. That is nasty.
>> Ok if that is imposable to extremely risky.
>>
>> What about a form of a proxy pid in the pid namespace proxying
>> application chatter between 1 name space to another. Applications

>> being the bridge if its not possible to do it invisible to application
>> could be made aware of it. So they can provide shared memory and the
>> like across pid namespaces. But only where they have a activated proxy
>> to do there bidding. This also allows applications to maintain there
>> own internal secuirty between namespaces.
>>
>> Ie application is 1 pid number in its source container and virtual pid
>> numbers in the following containers. Symbolic linking at task level
>> yes a little warped. Yes this will annoying mean a special set of
>> syscalls and a special set of capabilities and restrictions. Like PID
>> containers starting up forbidding proxy pid's or allowing them.
>>
>> If I am thinking right that avoids not be able to change it's pid.
>> Instead sending and receiving the messages you need in the other name
>> space threw a small proxy. Yes I know that will cost some
>> performance.
>
> Proxy pids don't actually do anything for you, unless you want to send
> signals. Because all of the namespaces are distinct. So even at the
> best of it you can see the X server but it still can't use your
> network sockets or ipc shm.
>
> Better is working out the details on how to manipulate multiple
> sysvipc and network namespaces from a single application. Mostly
> that is supported now by the objects there is just no easy way
> of dealing with it.
>
>> Basically want to setup a neat universal container way of handling
>> stuff like http://www.cs.toronto.edu/~andreslc/xen-gl/ without having
>> to go network and hopefully in a way that limitations don't have to
>> exist since messages are really only be sent threw 1 X11 server to 1
>> driver system. Only thing is really sending the correct messages to
>> the correct place. There will most likely be other services were a
>> single entity at times is preferred. Worst out come is if proxying
>> .so is required.
>
> Yes. I agree that is essentially desirable. Given that I think
> high end video card actually have multiple hardware contexts that
> can be mapped into different user space processes there may be other
> ways of handling this.
>
> Ideally we can find a high performance solution to X that also gives
> us good isolation and migration properties. Certainly something to talk
> about tomorrow in the conference.

In particular, if you wish to share private resources of a container
between more than a single container, then you won't be able to use
checkpoint/restart on neither container (unless you make special

provisions in the code).

I agree with Eric that the way to handle this is via virtualization
as opposed to direct sharing. The same goes for other hardware, e.g.
in the context of a user desktop - /dev/rtc, sound, and so on. My
experience is that a proxy/virtualized device is what we probably
want.

Oren.

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

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

---

Subject: Re: Roadmap for features planed for containers where and Some future features ideas.
Posted by Peter Dolding on Wed, 23 Jul 2008 00:56:46 GMT
View Forum Message <> Reply to Message

From: Kirill A. Shutemov <kirill@shutemov.name>

Changelog:
v4:
 - hierarchy support
 - drop dummy_timer_slack_check()
 - workaround lockdep false (?) positive
 - allow 0 as timer slack value
v3:
 - rework interface
 - s/EXPORT_SYMBOL/EXPORT_SYMBOL_GPL/
v2:
 - fixed with CONFIG_CGROUP_TIMER_SLACK=y
v1:
 - initial revision

Kirill A. Shutemov (2):
  cgroups: export cgroup_iter_{start,next,end}
  cgroups: introduce timer slack subsystem

```
 include/linux/cgroup_subsys.h |    6 +
 include/linux/init_task.h     |    4 +-
 init/Kconfig                  |   10 ++
 kernel/Makefile               |    1 +
 kernel/cgroup.c               |    3 +
 kernel/cgroup_timer_slack.c   |  262 ++++++++++++++++++++++++++++++++++++++++++
 kernel/sys.c                  |   19 ++-
 7 files changed, 298 insertions(+), 7 deletions(-)
 create mode 100644 kernel/cgroup_timer_slack.c

--
1.7.3.5


_____
Containers mailing list
Containers@lists.linux-foundation.org
 https://lists.linux-foundation.org/mailman/listinfo/containe rsFrom: Kirill A. Shutemov
<kirill@shutemov.name>

Signed-off-by: Kirill A. Shutemov <kirill@shutemov.name>
---
 kernel/cgroup.c |    3 +++
 1 files changed, 3 insertions(+), 0 deletions(-)

diff --git a/kernel/cgroup.c b/kernel/cgroup.c
index b24d702..8234daa 100644
--- a/kernel/cgroup.c
+++ b/kernel/cgroup.c
@@ -2443,6 +2443,7 @@ void cgroup_iter_start(struct cgroup *cgrp, struct cgroup_iter *it)
  it->cg_link = &cgrp->css_sets;
  cgroup_advance_iter(cgrp, it);
 }
+EXPORT_SYMBOL_GPL(cgroup_iter_start);

 struct task_struct *cgroup_iter_next(struct cgroup *cgrp,
      struct cgroup_iter *it)
@@ -2467,11 +2468,13 @@ struct task_struct *cgroup_iter_next(struct cgroup *cgrp,
  }
  return res;
 }
+EXPORT_SYMBOL_GPL(cgroup_iter_next);

 void cgroup_iter_end(struct cgroup *cgrp, struct cgroup_iter *it)
 {
  read_unlock(&css_set_lock);
 }
+EXPORT_SYMBOL_GPL(cgroup_iter_end);
```

```
 static inline int started_after_time(struct task_struct *t1,
        struct timespec *time,
--
1.7.3.5
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containe rsFrom: Kirill A. Shutemov <kirill@shutemov.name>

Provides a way of tasks grouping by timer slack value. Introduces per
cgroup max and min timer slack value. When a task attaches to a cgroup,
its timer slack value adjusts (if needed) to fit min-max range.

It also provides a way to set timer slack value for all tasks in the
cgroup at once.

This functionality is useful in mobile devices where certain background
apps are attached to a cgroup and minimum wakeups are desired.

Signed-off-by: Kirill A. Shutemov <kirill@shutemov.name>
Idea-by: Jacob Pan <jacob.jun.pan@linux.intel.com>
Signed-off-by: Kirill A. Shutemov <kirill@shutemov.name>

```
 include/linux/cgroup_subsys.h |    6 +
 include/linux/init_task.h     |    4 +-
 init/Kconfig                  |   10 ++
 kernel/Makefile               |    1 +
 kernel/cgroup_timer_slack.c   |  262 ++++++++++++++++++++++++++++++++++++++++++
 kernel/sys.c                  |   19 ++-
 6 files changed, 295 insertions(+), 7 deletions(-)
 create mode 100644 kernel/cgroup_timer_slack.c

diff --git a/include/linux/cgroup_subsys.h b/include/linux/cgroup_subsys.h
index ccefff0..e399228 100644
--- a/include/linux/cgroup_subsys.h
+++ b/include/linux/cgroup_subsys.h
@@ -66,3 +66,9 @@ SUBSYS(blkio)
 #endif

 /* */
+
+#ifdef CONFIG_CGROUP_TIMER_SLACK
+SUBSYS(timer_slack)
+#endif
+
+/* */
```

```
diff --git a/include/linux/init_task.h b/include/linux/init_task.h
index caa151f..48eca8f 100644
--- a/include/linux/init_task.h
+++ b/include/linux/init_task.h
@@ -124,6 +124,8 @@ extern struct cred init_cred;
 # define INIT_PERF_EVENTS(tsk)
 #endif

+#define TIMER_SLACK_NS_DEFAULT 50000
+
 /*
  *  INIT_TASK is used to set up the first task table, touch at
  * your own risk!. Base=0, limit=0x1fffff (=2MB)
@@ -177,7 +179,7 @@ extern struct cred init_cred;
  .cpu_timers = INIT_CPU_TIMERS(tsk.cpu_timers),  \
  .fs_excl = ATOMIC_INIT(0),    \
  .pi_lock = __RAW_SPIN_LOCK_UNLOCKED(tsk.pi_lock), \
- .timer_slack_ns = 50000, /* 50 usec default slack */  \
+ .timer_slack_ns = TIMER_SLACK_NS_DEFAULT,   \
  .pids = {       \
   [PIDTYPE_PID]  = INIT_PID_LINK(PIDTYPE_PID),  \
   [PIDTYPE_PGID] = INIT_PID_LINK(PIDTYPE_PGID),  \
diff --git a/init/Kconfig b/init/Kconfig
index be788c0..6cf465f 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -596,6 +596,16 @@ config CGROUP_FREEZER
    Provides a way to freeze and unfreeze all tasks in a
    cgroup.

+config CGROUP_TIMER_SLACK
+ tristate "Timer slack cgroup subsystem"
+ help
+   Provides a way of tasks grouping by timer slack value.
+   Introduces per cgroup timer slack value which will override
+   the default timer slack value once a task is attached to a
+   cgroup.
+   It's useful in mobile devices where certain background apps
+   are attached to a cgroup and combined wakeups are desired.
+
 config CGROUP_DEVICE
  bool "Device controller for cgroups"
  help
diff --git a/kernel/Makefile b/kernel/Makefile
index 353d3fe..0b60239 100644
--- a/kernel/Makefile
+++ b/kernel/Makefile
@@ -61,6 +61,7 @@ obj-$(CONFIG_BACKTRACE_SELF_TEST) += backtracetest.o
```

```
 obj-$(CONFIG_COMPAT) += compat.o
 obj-$(CONFIG_CGROUPS) += cgroup.o
 obj-$(CONFIG_CGROUP_FREEZER) += cgroup_freezer.o
+obj-$(CONFIG_CGROUP_TIMER_SLACK) += cgroup_timer_slack.o
 obj-$(CONFIG_CPUSETS) += cpuset.o
 obj-$(CONFIG_CGROUP_NS) += ns_cgroup.o
 obj-$(CONFIG_UTS_NS) += utsname.o
diff --git a/kernel/cgroup_timer_slack.c b/kernel/cgroup_timer_slack.c
new file mode 100644
index 0000000..affd33a
--- /dev/null
+++ b/kernel/cgroup_timer_slack.c
@@ -0,0 +1,262 @@
+/*
+ * cgroup_timer_slack.c - control group timer slack subsystem
+ *
+ * Copyright Nokia Corparation, 2011
+ * Author: Kirill A. Shutemov
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License as published by
+ * the Free Software Foundation; either version 2 of the License, or
+ * (at your option) any later version.
+ *
+ * This program is distributed in the hope that it will be useful,
+ * but WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
+ * GNU General Public License for more details.
+ */
+#include <linux/cgroup.h>
+#include <linux/init_task.h>
+#include <linux/module.h>
+#include <linux/slab.h>
+#include <linux/rcupdate.h>
+
+struct cgroup_subsys timer_slack_subsys;
+struct timer_slack_cgroup {
+ struct cgroup_subsys_state css;
+ unsigned long min_slack_ns;
+ unsigned long max_slack_ns;
+};
+
+enum {
+ TIMER_SLACK_MIN,
+ TIMER_SLACK_MAX,
+};
+
+extern int (*timer_slack_check)(struct task_struct *task,
```

```
+  unsigned long slack_ns);
+
+static struct timer_slack_cgroup *cgroup_to_tslack_cgroup(struct cgroup *cgroup)
+{
+ struct cgroup_subsys_state *css;
+
+ css = cgroup_subsys_state(cgroup, timer_slack_subsys.subsys_id);
+ return container_of(css, struct timer_slack_cgroup, css);
+}
+
+static int is_timer_slack_allowed(struct timer_slack_cgroup *t
```