
Subject: [RFC][PATCH 1/6] Add struct pid_nr
Posted by [Sukadev Bhattiprolu](#) on Sat, 10 Mar 2007 03:57:35 GMT
[View Forum Message](#) <> [Reply to Message](#)

From: Cedric Le Goater <clg@fr.ibm.com>
Subject: [RFC][PATCH 1/6] Add struct pid_nr

Define struct pid_nr and some helper functions that will be used in subsequent patches.

Changelog:

- [Serge Hallyn comment]: Remove (!pid_nr) check in free_pid_nr()

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>

include/linux/pid.h | 21 +++++
kernel/pid.c | 64 +++++
2 files changed, 85 insertions(+)

Index: lx26-20-mm2b/include/linux/pid.h

=====

--- lx26-20-mm2b.orig/include/linux/pid.h 2007-03-09 18:13:04.000000000 -0800

+++ lx26-20-mm2b/include/linux/pid.h 2007-03-09 18:27:51.000000000 -0800

@@ -11,6 +11,23 @@ enum pid_type

 PIDTYPE_MAX

};

+struct pid_namespace;

+

+/*

+ * A struct pid_nr holds a process identifier (or pid->nr) for a given

+ * pid namespace.

+ *

+ * A list of struct pid_nr is stored in the struct pid. this list is

+ * used to get the process identifier associated with the pid

+ * namespace it is being seen from.

+ */

+struct pid_nr

+{

+ struct hlist_node node;

+ int nr;

+ struct pid_namespace* pid_ns;

+};

+

/*

 * What is struct pid?

```

*
@@ -72,6 +89,7 @@ extern struct task_struct *FASTCALL(get_
    enum pid_type));

extern struct pid *get_task_pid(struct task_struct *task, enum pid_type type);
+extern void free_pidmap_pid_nr(struct pid_nr *);

/*
 * attach_pid() and detach_pid() must be called with the tasklist_lock
@@ -95,6 +113,9 @@ extern struct pid *FASTCALL(find_pid(int
extern struct pid *find_get_pid(int nr);
extern struct pid *find_ge_pid(int nr);

+extern int attach_pid_nr(struct pid *pid, struct pid_nr *pid_nr);
+extern void free_pid_nr(struct pid_nr *pid_nr);
+extern struct pid_nr *alloc_pid_nr(struct pid_namespace *pid_ns);
extern struct pid *alloc_pid(void);
extern void FASTCALL(free_pid(struct pid *pid));

```

Index: lx26-20-mm2b/kernel/pid.c

```

=====
--- lx26-20-mm2b.orig/kernel/pid.c 2007-03-09 18:13:04.000000000 -0800
+++ lx26-20-mm2b/kernel/pid.c 2007-03-09 18:27:51.000000000 -0800
@@ -33,6 +33,7 @@
static struct hlist_head *pid_hash;
static int pidhash_shift;
static struct kmem_cache *pid_cachep;
+static struct kmem_cache *pid_nr_cachep;
struct pid init_struct_pid = INIT_STRUCT_PID;

int pid_max = PID_MAX_DEFAULT;
@@ -190,6 +191,12 @@ static void delayed_put_pid(struct rcu_h
    put_pid(pid);
}

+void free_pid_nr(struct pid_nr *pid_nr)
+{
+    put_pid_ns(pid_nr->pid_ns);
+    kmem_cache_free(pid_nr_cachep, pid_nr);
+}
+
+fastcall void free_pid(struct pid *pid)
+{
+    /* We can be called with write_lock_irq(&tasklist_lock) held */
@@ -203,6 +210,59 @@ fastcall void free_pid(struct pid *pid)
    call_rcu(&pid->rcu, delayed_put_pid);
}

```

```

+struct pid_nr *alloc_pid_nr(struct pid_namespace *pid_ns)
+{
+ struct pid_nr* pid_nr;
+
+ pid_nr = kmem_cache_alloc(pid_nr_cachep, GFP_KERNEL);
+ if (!pid_nr)
+ return pid_nr;
+
+ INIT_HLIST_NODE(&pid_nr->node);
+ pid_nr->nr = -1;
+ get_pid_ns(pid_ns);
+ pid_nr->pid_ns = pid_ns;
+
+ return pid_nr;
+}
+
+void free_pidmap_pid_nr(struct pid_nr *pid_nr)
+{
+ free_pidmap(pid_nr->pid_ns, pid_nr->nr);
+ free_pid_nr(pid_nr);
+}
+
+static struct pid_nr *alloc_pidmap_pid_nr(struct pid_namespace *pid_ns)
+{
+ int nr;
+ struct pid_nr *pid_nr;
+
+ nr = alloc_pidmap(pid_ns);
+ if (nr < 0)
+ return NULL;
+
+ pid_nr = alloc_pid_nr(pid_ns);
+ if (!pid_nr)
+ goto out_free_pidmap;
+
+ pid_nr->nr = nr;
+
+ return pid_nr;
+
+out_free_pidmap:
+ free_pidmap(pid_ns, nr);
+
+ return NULL;
+}
+
+int attach_pid_nr(struct pid *pid, struct pid_nr *pid_nr)
+{
+ spin_lock(&pid->lock);

```

```

+ hlist_add_head_rcu(&pid_nr->node, &pid->pid_nrs);
+ spin_unlock(&pid->lock);
+ return 0;
+}
+
struct pid *alloc_pid(void)
{
    struct pid *pid;
@@ -420,4 +480,8 @@ void __init pidmap_init(void)
    pid_cachep = kmem_cache_create("pid", sizeof(struct pid),
        __alignof__(struct pid),
        SLAB_PANIC, NULL, NULL);
+
+ pid_nr_cachep = kmem_cache_create("pid_nr", sizeof(struct pid_nr),
+     __alignof__(struct pid_nr),
+     SLAB_PANIC, NULL, NULL);
}

```

Containers mailing list
Containers@lists.osdl.org
<https://lists.osdl.org/mailman/listinfo/containers>

Subject: Re: [RFC][PATCH 1/6] Add struct pid_nr
Posted by [ebiederm](#) on Sun, 11 Mar 2007 11:35:16 GMT
[View Forum Message](#) <> [Reply to Message](#)

sukadev@us.ibm.com writes:

```

> From: Cedric Le Goater <clg@fr.ibm.com>
> Subject: [RFC][PATCH 1/6] Add struct pid_nr
>
> Define struct pid_nr and some helper functions that will be used in
> subsequent patches.
>
> Changelog:
> - [Serge Halryn comment]: Remove (!pid_nr) check in free_pid_nr()
>
> Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
>
> ---
> include/linux/pid.h | 21 +++++
> kernel/pid.c | 64 +++++
> 2 files changed, 85 insertions(+)
>
> Index: lx26-20-mm2b/include/linux/pid.h
> =====

```

```

> --- lx26-20-mm2b.orig/include/linux/pid.h 2007-03-09 18:13:04.000000000 -0800
> +++ lx26-20-mm2b/include/linux/pid.h 2007-03-09 18:27:51.000000000 -0800
> @@ -11,6 +11,23 @@ enum pid_type
>  PIDTYPE_MAX
> };
>
> +struct pid_namespace;
> +
> +/*
> + * A struct pid_nr holds a process identifier (or pid->nr) for a given
> + * pid namespace.
> + *
> + * A list of struct pid_nr is stored in the struct pid. this list is
> + * used to get the process identifier associated with the pid
> + * namespace it is being seen from.
> + */
> +struct pid_nr
> +{
> + struct hlist_node node;
> + int nr;
> + struct pid_namespace* pid_ns;
> +};
> +
> +/*
> + * What is struct pid?
> + *
> @@ -72,6 +89,7 @@ extern struct task_struct *FASTCALL(get_
>  enum pid_type));
>
> extern struct pid *get_task_pid(struct task_struct *task, enum pid_type type);
> +extern void free_pidmap_pid_nr(struct pid_nr *);
>
> +/*
> + * attach_pid() and detach_pid() must be called with the tasklist_lock
> @@ -95,6 +113,9 @@ extern struct pid *FASTCALL(find_pid(int
> extern struct pid *find_get_pid(int nr);
> extern struct pid *find_ge_pid(int nr);
>
> +extern int attach_pid_nr(struct pid *pid, struct pid_nr *pid_nr);
> +extern void free_pid_nr(struct pid_nr *pid_nr);
> +extern struct pid_nr *alloc_pid_nr(struct pid_namespace *pid_ns);
> extern struct pid *alloc_pid(void);
> extern void FASTCALL(free_pid(struct pid *pid));
>
> Index: lx26-20-mm2b/kernel/pid.c
> =====
> --- lx26-20-mm2b.orig/kernel/pid.c 2007-03-09 18:13:04.000000000 -0800
> +++ lx26-20-mm2b/kernel/pid.c 2007-03-09 18:27:51.000000000 -0800

```

```

> @@ -33,6 +33,7 @@
> static struct hlist_head *pid_hash;
> static int pidhash_shift;
> static struct kmem_cache *pid_cachep;
> +static struct kmem_cache *pid_nr_cachep;
> struct pid init_struct_pid = INIT_STRUCT_PID;
>
> int pid_max = PID_MAX_DEFAULT;
> @@ -190,6 +191,12 @@ static void delayed_put_pid(struct rcu_h
> put_pid(pid);
> }
>
> +void free_pid_nr(struct pid_nr *pid_nr)
> +{
> + put_pid_ns(pid_nr->pid_ns);
> + kmem_cache_free(pid_nr_cachep, pid_nr);
> +}
> +
> fastcall void free_pid(struct pid *pid)
> {
> /* We can be called with write_lock_irq(&tasklist_lock) held */
> @@ -203,6 +210,59 @@ fastcall void free_pid(struct pid *pid)
> call_rcu(&pid->rcu, delayed_put_pid);
> }
>
> +struct pid_nr *alloc_pid_nr(struct pid_namespace *pid_ns)
> +{
> + struct pid_nr* pid_nr;
> +
> + pid_nr = kmem_cache_alloc(pid_nr_cachep, GFP_KERNEL);
> + if (!pid_nr)
> + return pid_nr;
> +
> + INIT_HLIST_NODE(&pid_nr->node);
> + pid_nr->nr = -1;
> + get_pid_ns(pid_ns);
> + pid_nr->pid_ns = pid_ns;
> +
> + return pid_nr;
> +}
> +
> +void free_pidmap_pid_nr(struct pid_nr *pid_nr)
> +{
> + free_pidmap(pid_nr->pid_ns, pid_nr->nr);
> + free_pid_nr(pid_nr);
> +}
> +
> +static struct pid_nr *alloc_pidmap_pid_nr(struct pid_namespace *pid_ns)

```

```

> +{
> +    int nr;
> +    struct pid_nr *pid_nr;
> +
> +    nr = alloc_pidmap(pid_ns);
> +    if (nr < 0)
> +        return NULL;
> +
> +    pid_nr = alloc_pid_nr(pid_ns);
> +    if (!pid_nr)
> +        goto out_free_pidmap;
> +
> +    pid_nr->nr = nr;
> +
> +    return pid_nr;
> +
> +out_free_pidmap:
> +    free_pidmap(pid_ns, nr);
> +
> +    return NULL;
> +}
> +
> +int attach_pid_nr(struct pid *pid, struct pid_nr *pid_nr)
> +{
> +    spin_lock(&pid->lock);
> +    hlist_add_head_rcu(&pid_nr->node, &pid->pid_nrs);
> +    spin_unlock(&pid->lock);

```

struct pid doesn't have a lock member.

We should be able to add everything to struct pid at allocation time, so we should not need a lock.

If you made struct pid_nr what the hash table entry it would probably make more sense, and gave it a struct pid pointer it would probably make more sense.

Eric

Containers mailing list
Containers@lists.osdl.org
<https://lists.osdl.org/mailman/listinfo/containers>

Subject: Re: [RFC][PATCH 1/6] Add struct pid_nr
Posted by [serue](#) on Sun, 11 Mar 2007 15:51:35 GMT
[View Forum Message](#) <> [Reply to Message](#)

Quoting Eric W. Biederman (ebiederm@xmission.com):

> sukadev@us.ibm.com writes:

>

>> From: Cedric Le Goater <clg@fr.ibm.com>

>> Subject: [RFC][PATCH 1/6] Add struct pid_nr

>>

>> Define struct pid_nr and some helper functions that will be used in

>> subsequent patches.

>>

>> Changelog:

>> - [Serge Hallyn comment]: Remove (!pid_nr) check in free_pid_nr()

>>

>> Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>

>> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>

>>

>> ---

>> include/linux/pid.h | 21 ++++++

>> kernel/pid.c | 64 ++++++

>> 2 files changed, 85 insertions(+)

>>

>> Index: lx26-20-mm2b/include/linux/pid.h

>> =====

>> --- lx26-20-mm2b.orig/include/linux/pid.h 2007-03-09 18:13:04.000000000 -0800

>> +++ lx26-20-mm2b/include/linux/pid.h 2007-03-09 18:27:51.000000000 -0800

>> @@ -11,6 +11,23 @@ enum pid_type

>> PIDTYPE_MAX

>> };

>>

>> +struct pid_namespace;

>> +

>> +/*

>> + * A struct pid_nr holds a process identifier (or pid->nr) for a given

>> + * pid namespace.

>> + *

>> + * A list of struct pid_nr is stored in the struct pid. this list is

>> + * used to get the process identifier associated with the pid

>> + * namespace it is being seen from.

>> + */

>> +struct pid_nr

>> +{

>> + struct hlist_node node;

>> + int nr;

>> + struct pid_namespace* pid_ns;

>> +};

>> +

>> /*

>> * What is struct pid?

>> *

```

>> @@ -72,6 +89,7 @@ extern struct task_struct *FASTCALL(get_
>>     enum pid_type));
>>
>> extern struct pid *get_task_pid(struct task_struct *task, enum pid_type type);
>> +extern void free_pidmap_pid_nr(struct pid_nr *);
>>
>> /*
>>  * attach_pid() and detach_pid() must be called with the tasklist_lock
>> @@ -95,6 +113,9 @@ extern struct pid *FASTCALL(find_pid(int
>> extern struct pid *find_get_pid(int nr);
>> extern struct pid *find_ge_pid(int nr);
>>
>> +extern int attach_pid_nr(struct pid *pid, struct pid_nr *pid_nr);
>> +extern void free_pid_nr(struct pid_nr *pid_nr);
>> +extern struct pid_nr *alloc_pid_nr(struct pid_namespace *pid_ns);
>> extern struct pid *alloc_pid(void);
>> extern void FASTCALL(free_pid(struct pid *pid));
>>
>> Index: lx26-20-mm2b/kernel/pid.c
>> =====
>> --- lx26-20-mm2b.orig/kernel/pid.c 2007-03-09 18:13:04.000000000 -0800
>> +++ lx26-20-mm2b/kernel/pid.c 2007-03-09 18:27:51.000000000 -0800
>> @@ -33,6 +33,7 @@
>> static struct hlist_head *pid_hash;
>> static int pidhash_shift;
>> static struct kmem_cache *pid_cachep;
>> +static struct kmem_cache *pid_nr_cachep;
>> struct pid init_struct_pid = INIT_STRUCT_PID;
>>
>> int pid_max = PID_MAX_DEFAULT;
>> @@ -190,6 +191,12 @@ static void delayed_put_pid(struct rcu_h
>> put_pid(pid);
>> }
>>
>> +void free_pid_nr(struct pid_nr *pid_nr)
>> +{
>> + put_pid_ns(pid_nr->pid_ns);
>> + kmem_cache_free(pid_nr_cachep, pid_nr);
>> +}
>> +
>> fastcall void free_pid(struct pid *pid)
>> {
>> /* We can be called with write_lock_irq(&tasklist_lock) held */
>> @@ -203,6 +210,59 @@ fastcall void free_pid(struct pid *pid)
>> call_rcu(&pid->rcu, delayed_put_pid);
>> }
>>
>> +struct pid_nr *alloc_pid_nr(struct pid_namespace *pid_ns)

```

```

>> +{
>> + struct pid_nr* pid_nr;
>> +
>> + pid_nr = kmem_cache_alloc(pid_nr_cachep, GFP_KERNEL);
>> + if (!pid_nr)
>> + return pid_nr;
>> +
>> + INIT_HLIST_NODE(&pid_nr->node);
>> + pid_nr->nr = -1;
>> + get_pid_ns(pid_ns);
>> + pid_nr->pid_ns = pid_ns;
>> +
>> + return pid_nr;
>> +}
>> +
>> +void free_pidmap_pid_nr(struct pid_nr *pid_nr)
>> +{
>> +    free_pidmap(pid_nr->pid_ns, pid_nr->nr);
>> +    free_pid_nr(pid_nr);
>> +}
>> +
>> +static struct pid_nr *alloc_pidmap_pid_nr(struct pid_namespace *pid_ns)
>> +{
>> +    int nr;
>> +    struct pid_nr *pid_nr;
>> +
>> +    nr = alloc_pidmap(pid_ns);
>> +    if (nr < 0)
>> +        return NULL;
>> +
>> +    pid_nr = alloc_pid_nr(pid_ns);
>> +    if (!pid_nr)
>> +        goto out_free_pidmap;
>> +
>> +    pid_nr->nr = nr;
>> +
>> +    return pid_nr;
>> +
>> +out_free_pidmap:
>> +    free_pidmap(pid_ns, nr);
>> +
>> +    return NULL;
>> +}
>> +
>> +int attach_pid_nr(struct pid *pid, struct pid_nr *pid_nr)
>> +{
>> +    spin_lock(&pid->lock);
>> +    hlist_add_head_rcu(&pid_nr->node, &pid->pid_nrs);

```

```
> > + spin_unlock(&pid->lock);
>
> struct pid doesn't have a lock member.
>
> We should be able to add everything to struct pid at allocation time,
> so we should not need a lock.
>
> If you made struct pid_nr what the hash table entry it would probably
> make more sense, and gave it a struct pid pointer it would probably
> make more sense.
```

i guess this is one reason to not support unshare for pidns. if we only support clone thenj we don't need the lock.

actually that's not true. whenever we start to allow clone pidns without doing setsid first, we'll need to pull the existing processes which are session and prgp leaders into the new pidns, so we will need to add pidnr's to their struct pid.

-serge

Containers mailing list
Containers@lists.osdl.org
<https://lists.osdl.org/mailman/listinfo/containers>

Subject: Re: [RFC][PATCH 1/6] Add struct pid_nr
Posted by [ebiederm](#) on Sun, 11 Mar 2007 18:13:47 GMT
[View Forum Message](#) <> [Reply to Message](#)

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

```
>> > +int attach_pid_nr(struct pid *pid, struct pid_nr *pid_nr)
>> > +{
>> > + spin_lock(&pid->lock);
>> > + hlist_add_head_rcu(&pid_nr->node, &pid->pid_nrs);
>> > + spin_unlock(&pid->lock);
>>
>> struct pid doesn't have a lock member.
>>
>> We should be able to add everything to struct pid at allocation time,
>> so we should not need a lock.
>>
>> If you made struct pid_nr what the hash table entry it would probably
>> make more sense, and gave it a struct pid pointer it would probably
>> make more sense.
>
> i guess this is one reason to not support unshare for pidns. if we only
```

> support clone thenj we don't need the lock.
>
> actually that's not true. whenever we start to allow clone pidns
> without doing setsid first, we'll need to pull the existing processes
> which are session and prgp leaders into the new pidns, so we will need
> to add pidnr's to their struct pid.

A better way to put this is that we already have a lock that attach_pid and detach_pid use. So we don't need another one for what should be a very rare case.

I don't think we will need to add pids in the new pid namespace for sid and prgp leaders into the new pid namespace.

If we do need to pull in sid and prgp leaders calling setsid() before the operation won't help (wrong namespace) and the problem is the same for unshare and clone.

Eric

Containers mailing list
Containers@lists.osdl.org
<https://lists.osdl.org/mailman/listinfo/containers>

Subject: Re: [RFC][PATCH 1/6] Add struct pid_nr
Posted by [serue](#) on Sun, 11 Mar 2007 20:38:30 GMT
[View Forum Message](#) <> [Reply to Message](#)

Quoting Eric W. Biederman (ebiederm@xmission.com):

> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>
> >> +int attach_pid_nr(struct pid *pid, struct pid_nr *pid_nr)
> >> +{
> >> + spin_lock(&pid->lock);
> >> + hlist_add_head_rcu(&pid_nr->node, &pid->pid_nrs);
> >> + spin_unlock(&pid->lock);
> >>
> >> struct pid doesn't have a lock member.
> >>
> >> We should be able to add everything to struct pid at allocation time,
> >> so we should not need a lock.
> >>
> >> If you made struct pid_nr what the hash table entry it would probably
> >> make more sense, and gave it a struct pid pointer it would probably
> >> make more sense.
> >
> > i guess this is one reason to not support unshare for pidns. if we only

> > support clone thenj we don't need the lock.
> >
> > actually that's not true. whenever we start to allow clone pidns
> > without doing setsid first, we'll need to pull the existing processes
> > which are session and prgp leaders into the new pidns, so we will need
> > to add pidnr's to their struct pid.
>
> A better way to put this is that we already have a lock that attach_pid
> and detach_pid use. So we don't need another one for what should be
> a very rare case.

I think we'd at least need rcu if we supported unshare.

> I don't think we will need to add pids in the new pid namespace for
> sid and pgrp leaders into the new pid namespace.
>
> If we do need to pull in sid and pgrp leaders calling setsid() before
> the operation won't help (wrong namespace)

I'm pretty sure sid and pgrp leaders are a single task_struct pointer each, so setting these to point to yourself before an unshare works. But I guess for clone it does not work!

I agree with what you say in a later email - just returning '0' if sid or pgrp is not in our pid_ns presents no problems, and not having a pgrp on which to do kill -pgrp doesn't matter either...

So how about we

1. remove the setsid requirement before CLONE_NEWPID
2. punt on unshare(CLONE_NEWPID) support for now
3. remove pid->lock saving some space and time for now

> and the problem is the same
> for unshare and clone.

Disagree, but irrelevant if we do the above for now.

-serge

Containers mailing list
Containers@lists.osdl.org
<https://lists.osdl.org/mailman/listinfo/containers>

Subject: Re: [RFC][PATCH 1/6] Add struct pid_nr
Posted by [ebiederm](#) on Sun, 11 Mar 2007 22:05:03 GMT

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

>> A better way to put this is that we already have a lock that attach_pid
>> and detach_pid use. So we don't need another one for what should be
>> a very rare case.

>
> I think we'd at least need rcu if we supported unshare.

We might.

>> I don't think we will need to add pids in the new pid namespace for
>> sid and pgrp leaders into the new pid namespace.

>>
>> If we do need to pull in sid and pgrp leaders calling setsid() before
>> the operation won't help (wrong namespace)

>
> I'm pretty sure sid and pgrp leaders are a single task_struct pointer
> each, so setting these to point to yourself before an unshare works.
> But I guess for clone it does not work!

Point.

> I agree with what you say in a later email - just returning '0' if sid or
> pgrp is not in our pid_ns presents no problems, and not having a pgrp on
> which to do kill -pgrp doesn't matter either...

>
> So how about we
>
> 1. remove the setsid requirement before CLONE_NEWPID
> 2. punt on unshare(CLONE_NEWPID) support for now
punt as in skip it for the moment (I agree).

> 3. remove pid->lock saving some space and time for now

Sounds good.

>> and the problem is the same
>> for unshare and clone.

>
> Disagree, but irrelevant if we do the above for now.

I see your point about unshare being a little simpler because we do
allocate the local pids before we call unshare.

Eric

Containers mailing list

