Subject: kernel thread accounted to a VE
Posted by Eric Keller on Tue, 11 Dec 2007 17:11:09 GMT
View Forum Message <> Reply to Message

Is it possible to start a kernel thread and then move it to a particular VE?

I have the following code inside of a kernel thread:
envid_t _veid = 200;
// enter that VE
unsigned flags = VE_ENTER;
int err = real_env_create(_veid, flags, 0, 0, 0); // the last 3 arguments are only used if flags is VE_CREATE

I needed to modify ve_move_task() a bit. It has the following assignment: tsk->mm->vps_dumpable = 0; But for kernel_threads, tsk->mm is NULL, so I just check if it's null and don't do the assignment if it is null. Other than that, it appears to be successful. It returns successful and in the VE I moved the task to, I can see a new process running (using top).

The problem is, I set a cpu limit for that VE to 10%, yet I can see this thread go well above that amount (~50%). User processes do get limited when I run them, so I know it's not a setting issue (unless there's something special I need to do for kernel threads). Note that I do not want to allow the VEs to install kernel modules, so I want the host system to do it on their behalf for a very specific circumstance.

Any ideas of what I'm doing wrong or what it'll take to make this work?

Thanks,
Eric

Subject: Re: kernel thread accounted to a VE
Posted by den on Wed, 12 Dec 2007 07:36:02 GMT
View Forum Message <> Reply to Message

Eric Keller wrote:
> Is it possible to start a kernel thread and then move it to a particular
> VE?
> I have the following code inside of a kernel thread:
> envid_t _veid = 200;
> // enter that VE
> unsigned flags = VE_ENTER;
> int err = real_env_create(_veid, flags, 0, 0, 0); // the last 3
> arguments are only used if flags is VE_CREATE
>

> I needed to modify ve_move_task() a bit. It has the following
> assignment: tsk->mm->vps_dumpable = 0;   But for kernel_threads,
> tsk->mm is NULL, so I just check if it's null and don't do the
> assignment if it is null. Other than that, it appears to be
> successful. It returns successful and in the VE I moved the task to, I
> can see a new process running (using top).
>
> The problem is, I set a cpu limit for that VE to 10%, yet I can see this
> thread go well above that amount (~50%). User processes do get limited
> when I run them, so I know it's not a setting issue (unless there's
> something special I need to do for kernel threads). Note that I do not
> want to allow the VEs to install kernel modules, so I want the host
> system to do it on their behalf for a very specific circumstance.
>
> Any ideas of what I'm doing wrong or what it'll take to make this work?

first of all, you should check that enter was successful :) The most
simple case is that is don't. This can be confirmed by the ret code
checking and via /proc/<pid>/status of the particular thread

Regards,
 Den

Subject: Re: kernel thread accounted to a VE
Posted by dev on Wed, 12 Dec 2007 08:35:10 GMT
View Forum Message <> Reply to Message

Eric Keller wrote:
> Is it possible to start a kernel thread and then move it to a particular
> VE?
>
> I have the following code inside of a kernel thread:
> envid_t _veid = 200;
> // enter that VE
> unsigned flags = VE_ENTER;
> int err = real_env_create(_veid, flags, 0, 0, 0); // the last 3
> arguments are only used if flags is VE_CREATE
>
> I needed to modify ve_move_task() a bit. It has the following
> assignment: tsk->mm->vps_dumpable = 0;   But for kernel_threads,
> tsk->mm is NULL, so I just check if it's null and don't do the
> assignment if it is null. Other than that, it appears to be
> successful. It returns successful and in the VE I moved the task to, I
> can see a new process running (using top).
>
> The problem is, I set a cpu limit for that VE to 10%, yet I can see this
> thread go well above that amount (~50%). User processes do get limited

> when I run them, so I know it's not a setting issue (unless there's
> something special I need to do for kernel threads). Note that I do not
> want to allow the VEs to install kernel modules, so I want the host
> system to do it on their behalf for a very specific circumstance.
>
> Any ideas of what I'm doing wrong or what it'll take to make this work?

you can fix the place about checking for tsk->mm != NULL.
But... plz keep in mind the following:
1. having a kernel thread inside VE will break checkpointing (live migration),
   since CPT doesn't know how to restore this thread.
   (it can be fixed by you if you know how to save/restore it's state).
2. your kernel thread should handle signals or have an ability
   to detect VE shutdown, otherwise it will block VE stop.

and maybe something else...

Thanks,
Kirill

Subject: Re: kernel thread accounted to a VE
Posted by Eric Keller on Wed, 12 Dec 2007 16:11:31 GMT
View Forum Message <> Reply to Message

>
> first of all, you should check that enter was successful :) The most
> simple case is that is don't. This can be confirmed by the ret code
> checking and via /proc/<pid>/status of the particular thread
>

The return code says it was successful. And I performed 3 commands to
further check:
[#HN#] top
[#HN#]$ more /proc/12784/status
[#VE200#] top

The results are below, making it appear to be successful. What else
can I try or what debugging flags are there to see info about the
scheduler or where in the code does the cpu limit get enforced (and do
kernel threads get checked in that code)... or what are the right
questions for me to ask you guys?

Thanks for your help,
Eric

[#HN#] top

```
top - 10:39:27 up 7 min,  3 users,  load average: 0.65, 0.52, 0.26
Tasks: 118 total,   3 running, 115 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.7% us, 28.2% sy,  0.0% ni, 71.0% id,  0.0% wa,  0.0% hi,  0.2% si
Mem:   1989596k total,   422992k used,  1566604k free,    42488k buffers
Swap:  4192924k total,        0k used,  4192924k free,   210524k cached
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
12784 root      15   0     0    0    0 R   56  0.0   0:28.56 kclick


[#HN#]$ more /proc/12784/status
Name:   kclick
State:  R (running)
SleepAVG:       98%
Tgid:   12784
Pid:    12784
PPid:   1
TracerPid:      0
FNid:   200
Uid:    0       0       0       0
Gid:    0       0       0       0
FDSize: 64
Groups: 0 1 2 3 4 6 10
envID:  200
VPid:   13808
PNState:        0
StopState:      0
Threads:        1

[#VE200#] top
Tasks:  20 total,   2 running,  18 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0% us, 27.8% sy,  0.0% ni, 72.2% id,  0.0% wa,  0.0% hi,  0.0% si
Mem:    140000k total,    10640k used,   129360k free,        0k buffers
Swap:        0k total,        0k used,        0k free,        0k cached
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
13808 root      15   0     0    0    0 S   56  0.0   1:33.67 kclick
```