Subject: Which of the virtualization approaches is more suitable for kernel?
Posted by dev on Mon, 20 Feb 2006 15:42:28 GMT
View Forum Message <> Reply to Message

Linus, Andrew,

We need your help on what virtualization approach you would accept to mainstream (if any) and where we should go.

If to drop VPID virtualization which caused many disputes, we actually have the one virtualization solution, but 2 approaches for it. Which one will go depends on the goals and your approval any way.

So what are the approaches?

1. namespaces for all types of resources (Eric W. Biederman)

The proposed solution is similar to fs namespaces. This approach introduces namespaces for IPC, IPv4 networking, IPv6 networking, pids etc. It also adds to task_struct a set of pointers to namespaces which task belongs to, i.e. current->ipv4_ns, current->ipc_ns etc.

Benefits:
- fine grained namespaces
- task_struct points directly to a structure describing the namespace and it's data (not to container)
- maybe a bit more logical/clean implementation, since no effective container is involved unlike a second approach.

Disadvantages:
- it is only proof of concept code right now. nothing more.
- such an approach requires adding of additional argument to many functions (e.g. Eric's patch for networking is 1.5 bigger than openvz). it also adds code for getting namespace from objects. e.g. skb->sk->namespace.
- so it increases stack usage
- it can't efficiently compile to the same not virtualized kernel, which can be undesired for embedded linux.
- fine grained namespaces are actually an obfuscation, since kernel subsystems are tightly interconnected. e.g. network -> sysctl -> proc, mqueues -> netlink, ipc -> fs and most often can be used only as a whole container.
- it involves a bit more complicated procedure of a container create/enter which requires exec or something like this, since there is no effective container which could be simply triggered.

2. containers (OpenVZ.org/linux-vserver.org)

Container solution was discussed before, and actually it is also
namespace solution, but as a whole total namespace, with a single kernel
structure describing it.
Every task has two cotnainer pointers: container and effective
container. The later is used to temporarily switch to other contexts,
e.g. when handling IRQs, TCP/IP etc.

Benefits:
- clear logical bounded container, it is clear when container is alive
and when not.
- it doesn't introduce additional args for most functions, no additional
stack usage.
- it compiles to old good kernel when virtualization if off, so doesn't
disturb other configurations.
- Eric brought an interesting idea about introducing interface like
DEFINE_CPU_VAR(), which could potentially allow to create virtualized
variables automagically and access them via econtainer().
- mature working code exists which is used in production for years, so
first working version can be done much quicker

Disadvantages:
- one additional pointer dereference when accessing virtualized
resources, e.g. current->econtainer->shm_ids

Kirill

---