Introduction to

Linux Containers

Very briefly, **LXC** is a Linux kernel facility which allows processes to be “sandboxed” or isolated from each other. It internally employs these built-in kernel features – namespaces and Cgroups.

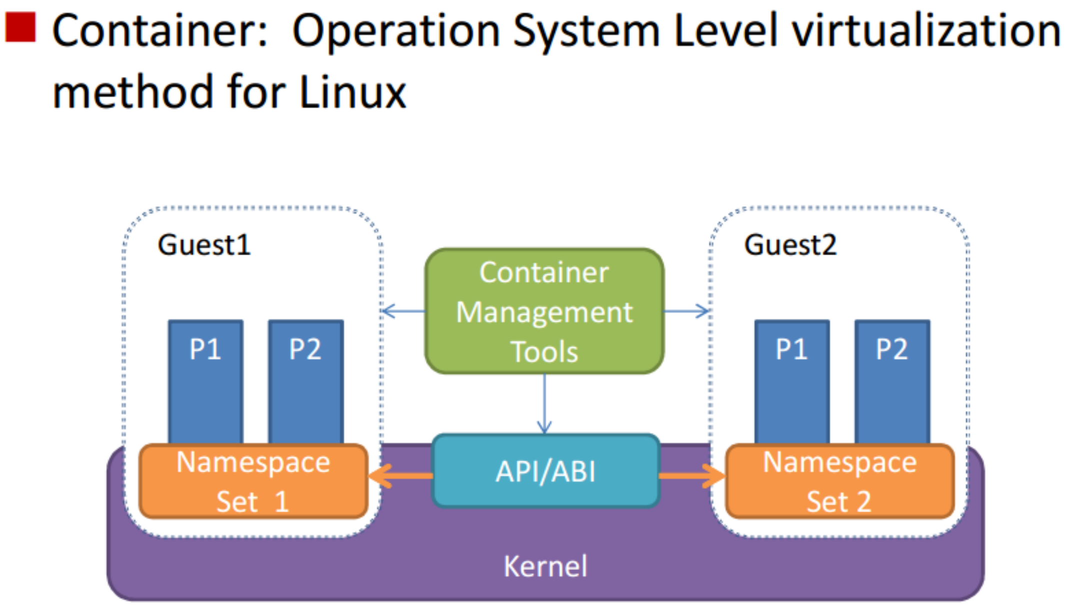


Diagram [Source: “LXC: Lightweight virtual system mechanism”, Gao feng]

[Source]

... What it basically does is isolate applications from others. A bit like chroot does by isolating applications in a virtual private root but taking the process further.

Internally, LXC relies on 3 main isolation infrastructure of the Linux Kernel:

1. Chroot
2. [Cgroups]

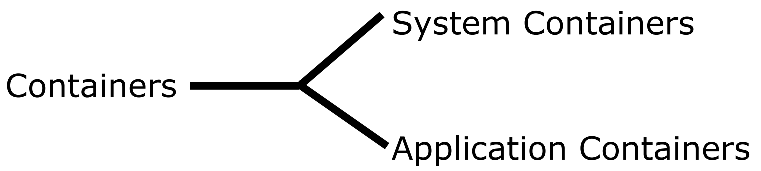
3. Namespaces
…

[Source]

LXC (**Linux Containers**) is an operating system–level virtualization method for running multiple isolated Linux systems (containers) on a single control host.

The Linux kernel comprises cgroups for resource isolation (CPU, memory, block I/O, network, etc.) that does not require starting any virtual machines. Cgroups also provides namespace isolation to completely isolate applications' view of the operating environment, including process trees, network, user ids and mounted file systems.

LXC combines cgroups and namespace support to provide an isolated environment for applications. Docker can also use LXC as one of its execution drivers, enabling image management and providing deployment services.



“LXC's main focus is system containers. That is, containers which offer an environment as close as possible as the one you'd get from a VM but without the overhead that comes with running a separate kernel and simulating all the hardware. ...”

Source: https://linuxcontainers.org/
“… LXC is often considered as something in the middle between a chroot on steroids and a full fledged virtual machine. The goal of LXC is to create an environment as close as possible as a standard Linux installation but without the need for a separate kernel. …”

Another thing: LXC requires a Linux kernel. If one is not on Linux, one can always run a Linux VM and run several Linux Containers (a la LXC) *within* it!

<<

Also see:

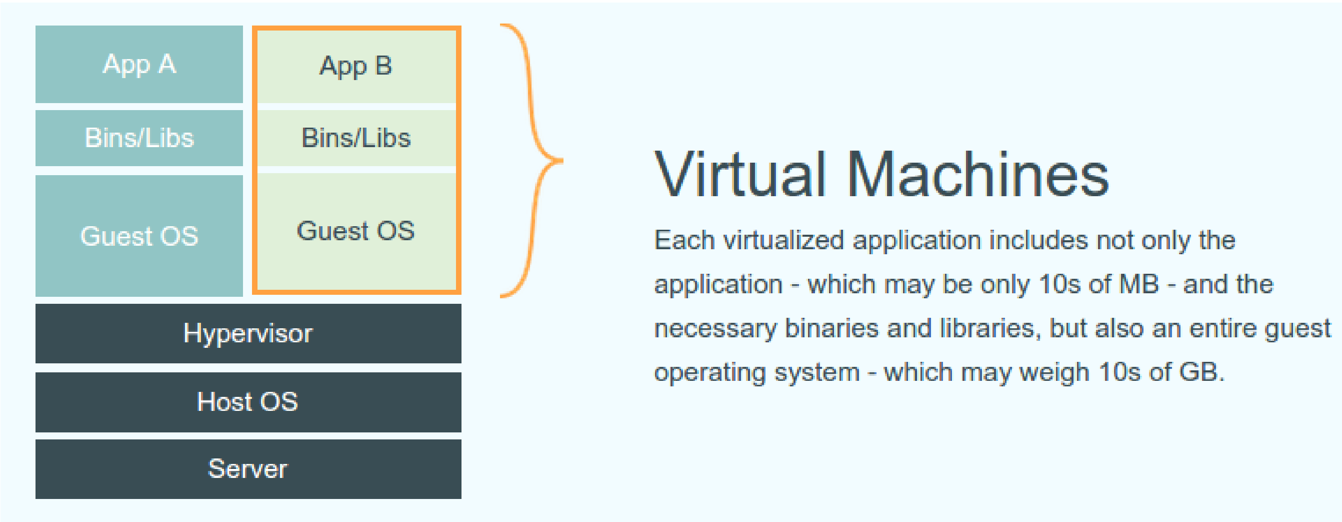
- systemd's *systemd-nspawn* command: “... systemd-nspawn may be used to run a command or OS in a light-weight namespace container. In many ways it is similar to chroot(1), but more powerful since it fully virtualizes the file system hierarchy, as well as the process tree, the various IPC subsystems and the host and domain name. …”

(Above snippet is from the man page of systemd-nspawn; see the examples!).
Note- systemd-nspawn, machinectl, etc requires the *systemd-container* package to be installed (on Ubuntu/Debian).

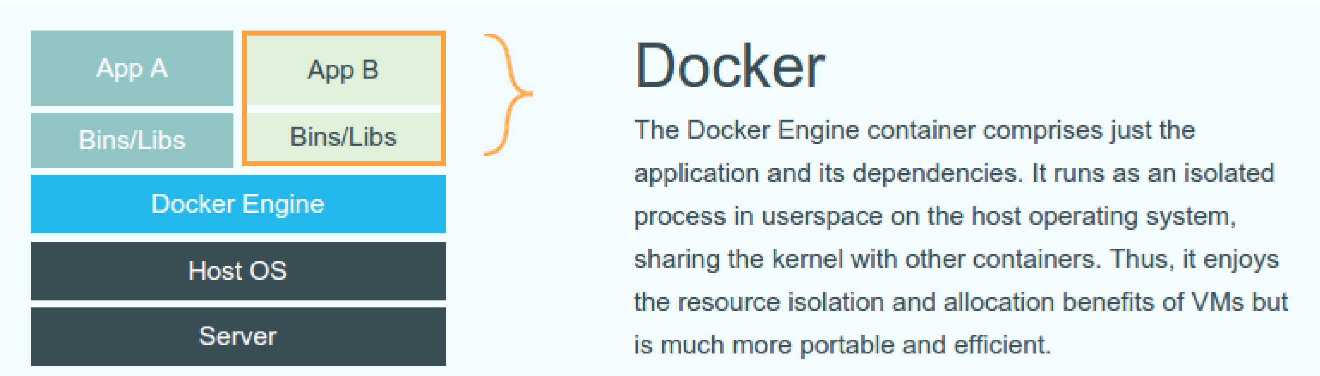
- User namespace support completed [3.8] [LWN articles refs here too]
- Today's (2016-17) realistic container solutions for the Linux *desktop*- Flatpak and Snap.

>>

From docker.com :



How is this different from Virtual Machines?



<< BTW, *"How to Install and Use Docker on Linux", linux.com* >>

So, briefly, some pros and cons:

Containers	***Virtualization***
Same host OS is shared => Performance benefit	Each VM has it's own kernel and each instruction has to be emulated => Performance hit
New-ish and requires a Linux host	All major OS's have mature virtualization solutions (hypervisors)
Less flexible; Linux-only solution	Flexible; different guest OS's can run in parallel
Different Linux distro on same CPU can be run in a container	Pretty much any OS on several CPU choices (via QEMU)
Migration is less flexible	Migration is more or less assured (with major vendors following standards like the Open Virtualization Format)
Security : regular	Robust

Source: "LXC: Lightweight virtual system mechanism", Gao feng

	Container	KVM
OS support	Linux Only	No Limit
Completeness	Low	Great
Security	Normal	Great
performance	Great	Normal

Detailed Benchmark and Analysis: see this IBM Research document - "An Updated Performance Comparison of Virtual Machines and Linux Containers".

A nice introduction:

"Condensing Your Infrastructure with System Containers

By Swapnil Bhartiya

Canonical's Stéphane Graber explains the difference between system and application containers in this OS Summit preview."

YOU WANT TO BUILD AN EMPIRE LIKE GOOGLE'S? THIS IS YOUR OS, Wired, Apr 2016

"… The move comes amid an enormous revolution sweeping information technology, one in which big-name companies and startups alike aim to recreate Borg for the rest of the world. Alex Polvi, who runs one of these startups, CoreOS, describes the revolution with a hashtag: **#GIFEE, or Google Infrastructure For Everyone Else**—

which is even catchier. In addition to Mesosphere and CoreOS, a company called Docker is pushing this idea alongside the biggest names in cloud computing: Amazon, Microsoft, and, yes, Google. ...

...

In the summer of 2014, the company << *Google* >> unveiled **Kubernetes**, its own open source effort to create a version of Borg others could use. Now that Kubernetes is open source, it seems, Mesosphere must open source all of DC/OS. By itself, Mesos provides only part of what Kubernetes offers.

...”

Creating an LXC Container

Discussion below is specific to LXC (pronounced "lex-see") – LinuX Containers – on Ubuntu. LXC is an early and popular container technology.
Reference site: Ubuntu Documentation: LXC

Firstly, if not already present, install the 'lxc' package:

```
# sudo apt-get install lxc
...

# lxc-create -n u1
A template must be specified.
Use "none" if you really want a container without a rootfs.

#
```

What "templates" are available?
"...
Creating a container generally involves creating a root filesystem for the container. `lxc-create` delegates this work to *templates*, which are generally per-distribution. The lxc templates shipped with lxc can be found under `/usr/share/lxc/templates`, and include templates to create Ubuntu, Debian, Fedora, Oracle, centos, and gentoo containers among others.

Creating distribution images in most cases requires the ability to create device nodes, often requires tools which are not available in other distributions, and usually is quite time-consuming. Therefore lxc comes with a special *download* template, which downloads pre-built container images from a central lxc server. The most important use case is to allow simple creation of unprivileged containers by non-root users, who could not for instance easily run the `debootstrap` command.
..."

```
$ ls /usr/share/lxc/templates/
lxc-alpine*    lxc-archlinux*  lxc-centos*  lxc-debian*    lxc-fedora*  lxc-
openmandriva*  lxc-oracle*     lxc-sshd*    lxc-ubuntu-cloud*
lxc-altlinux*  lxc-busybox*    lxc-cirros*  lxc-download*  lxc-gentoo*  lxc-
opensuse*      lxc-plamo*     lxc-ubuntu*
$
```

For the following operations, we require to run as superuser:

```
$ sudo /bin/bash
[sudo] password for <...>:
#
# lxc-create -n u1 --template ubuntu-cloud                   << 206 MB download >>
ubuntu-cloudimg-query is /usr/bin/ubuntu-cloudimg-query
wget is /usr/bin/wget
--2015-08-20 13:37:22--  https://cloud-
```

```
images.ubuntu.com/server/releases/vivid/release-20150818/ubuntu-15.04-server-
cloudimg-amd64-root.tar.gz
Resolving cloud-images.ubuntu.com (cloud-images.ubuntu.com)... 91.189.88.141,
2001:67c:1360:8001:ffff:ffff:ffff:fffe
Connecting to cloud-images.ubuntu.com (cloud-images.ubuntu.com)|
91.189.88.141|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://cloud-images.ubuntu.com/releases/vivid/release-
20150818/ubuntu-15.04-server-cloudimg-amd64-root.tar.gz [following]
--2015-08-20 13:37:23--  https://cloud-images.ubuntu.com/releases/vivid/release-
20150818/ubuntu-15.04-server-cloudimg-amd64-root.tar.gz
Reusing existing connection to cloud-images.ubuntu.com:443.
HTTP request sent, awaiting response... 200 OK
Length: 215577786 (206M) [application/x-gzip]
Saving to: ‘ubuntu-15.04-server-cloudimg-amd64-root.tar.gz’

ubuntu-15.04-server-cloudimg-amd64-ro 100%
[=============================================================================>]
205.59M  2.58MB/s   in 98s

2015-08-20 13:39:01 (2.10 MB/s) - ‘ubuntu-15.04-server-cloudimg-amd64-
root.tar.gz’ saved [215577786/215577786]

Extracting container rootfs
perl: warning: Setting locale failed.
perl: warning: Please check that your locale settings:
      LANGUAGE = "en_IN:en",
      LC_ALL = (unset),
      LANG = "en_IN"
    are supported and installed on your system.
perl: warning: Falling back to the standard locale ("C").
locale: Cannot set LC_CTYPE to default locale: No such file or directory
locale: Cannot set LC_MESSAGES to default locale: No such file or directory
locale: Cannot set LC_ALL to default locale: No such file or directory

Current default time zone: 'Asia/Kolkata'
Local time is now:      Thu Aug 20 13:39:09 IST 2015.
Universal Time is now:  Thu Aug 20 08:09:09 UTC 2015.

Container u1 created.
# ls -l /var/lib/lxc
total 4
drwxrwx--- 3 root root 4096 Aug 20 13:39 u1/
# ls -l /var/lib/lxc/u1/
total 8
-rw-r--r--  1 root root  519 Aug 20 13:39 config
drwxr-xr-x 22 root root 4096 Aug 20 13:39 rootfs/
# ls -l /var/lib/lxc/u1/rootfs/
total 80
drwxr-xr-x  2 root root 4096 Aug 18 12:51 bin/
drwxr-xr-x  3 root root 4096 Aug 18 12:56 boot/
drwxr-xr-x  4 root root 4096 Aug 20 13:39 dev/
drwxr-xr-x 91 root root 4096 Aug 20 13:39 etc/
drwxr-xr-x  2 root root 4096 Apr 18 03:04 home/
drwxr-xr-x 20 root root 4096 Aug 18 12:56 lib/
drwxr-xr-x  2 root root 4096 Aug 18 12:49 lib64/
drwx------  2 root root 4096 Aug 18 12:52 lost+found/
drwxr-xr-x  2 root root 4096 Aug 18 12:49 media/
```

```
drwxr-xr-x  2 root root 4096 Apr 18 03:04 mnt/
drwxr-xr-x  2 root root 4096 Aug 18 12:49 opt/
drwxr-xr-x  2 root root 4096 Apr 18 03:04 proc/
drwx------  2 root root 4096 Aug 18 12:51 root/
drwxr-xr-x  2 root root 4096 Aug 18 12:51 run/
drwxr-xr-x  2 root root 4096 Aug 18 12:51 sbin/
drwxr-xr-x  2 root root 4096 Aug 18 12:49 srv/
drwxr-xr-x  2 root root 4096 Apr  7 00:15 sys/
drwxrwxrwt  2 root root 4096 Aug 18 12:56 tmp/
drwxr-xr-x 10 root root 4096 Aug 18 12:49 usr/
drwxr-xr-x 12 root root 4096 Aug 18 12:51 var/
# du -ms /var/lib/lxc/u1/rootfs/
652   /var/lib/lxc/u1/rootfs/       << rootfs ~ 650 MB >>
#


# lxc-ls
u1
# lxc-ls --fancy
NAME  STATE    IPV4  IPV6  GROUPS  AUTOSTART
--------------------------------------------
u1    STOPPED  -     -     -       NO
# lxc-info -n u1
Name:           u1
State:          STOPPED
#
# lxc-start -n u1   << Start the container >>
#

# lxc-info -n u1
Name:           u1
State:          RUNNING
PID:            13151           << see PID investigation below >>
IP:             10.0.3.87
CPU use:        4.63 seconds
BlkIO use:      33.50 MiB
Memory use:     55.74 MiB
KMem use:       0 bytes
Link:           vethQHQA61
 TX bytes:      5.20 KiB
 RX bytes:      14.47 KiB
 Total bytes:   19.68 KiB
#
```

Enter the Container using lxc-attach

```
# lxc-attach -n u1
root@u1:/# ls       << we're now in a shell within the container! >>
bin  boot  dev   etc  home  lib   lib64  lost+found  media  mnt  opt  proc
root run  sbin  srv    sys  tmp  usr  var
root@u1:/# pwd
/
root@u1:/#
```

<<
On the “host” system:

```
$ ps -A|grep 13151
13151 ?        00:00:00 systemd << now a new 'systemd' process (note it's pid
is not 1) – this corresponds to the LXC “root” pid 1 process within the
container; the root of the container process hierachy! >>
$ ps -Al|head -1
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
$ ps -Al|grep 13151
4 S     0 13151 13138  0  80   0 -  8732 ep_pol ?        00:00:00 systemd
4 S     0 13917 13151  0  80   0 -  8283 ep_pol ?        00:00:00 systemd-
journal
1 S     0 14540 13151  0  80   0 -  5872 poll_s ?        00:00:00 dhclient
4 S     0 14669 13151  0  80   0 -  6513 hrtime ?        00:00:00 cron
4 S   104 14737 13151  0  80   0 - 48649 poll_s ?        00:00:00 rsyslogd
4 S     0 14970 13151  0  80   0 -  5007 ep_pol ?        00:00:00 systemd-logind
5 S     0 15044 13151  0  80   0 -  4878 hrtime ?        00:00:00 irqbalance
4 S     0 15045 13151  0  80   0 - 70429 poll_s ?        00:00:00 accounts-
daemon
4 S     1 15082 13151  0  80   0 -  4795 hrtime ?        00:00:00 atd
4 S   105 15253 13151  0  80   0 - 10592 ep_pol ?        00:00:00 dbus-daemon
4 S     0 15400 13151  0  80   0 - 69671 poll_s ?        00:00:00 polkitd
4 S     0 15740 13151  0  80   0 - 14910 poll_s ?        00:00:00 sshd
4 S     0 15993 13151  0  80   0 -  3206 wait_w pts/1    00:00:00 agetty
4 S     0 16013 13151  0  80   0 -  3206 wait_w pts/1    00:00:00 agetty
4 S     0 16033 13151  0  80   0 -  3206 wait_w pts/2    00:00:00 agetty
4 S     0 16053 13151  0  80   0 -  3206 wait_w pts/3    00:00:00 agetty
4 S     0 16073 13151  0  80   0 -  3206 wait_w pts/0    00:00:00 agetty
$
```

Another (better!) way of seeing this is via ***pstree***

```
# pstree --highlight-pid=13151 --show-pids
...
        |          |          |           ├─knotify4(29863)─┬─{QInotifyFileSys}(29868)
        |          |          |           |                 ├─{gdbus}(29864)
        |          |          |           |                 └─{gmain}(29865)
        |          |          |           ├─lxc-start(13138)───systemd(13151)─┬─accounts-
daemon(15045)─┬─{gdbus}(15274)
        |          |          |           |                      |
└─{gmain}(15079)
        |          |          |           |                      ├─agetty(15993)
        |          |          |           |                      ├─agetty(16013)
        |          |          |           |                      ├─agetty(16033)
        |          |          |           |                      ├─agetty(16053)
        |          |          |           |                      ├─agetty(16073)
        |          |          |           |                      ├─atd(15082)
        |          |          |           |                      ├─cron(14669)
        |          |          |           |                      ├─dbus-
daemon(15253)
        |          |          |           |
├─dhclient(14540)
        |          |          |           |
├─irqbalance(15044)
        |          |          |           |
```

```
├polkitd(15400)─┬{gdbus}(15437)
     │         │         │          │                      │
└{gmain}(15440)
     │         │         │          │
├rsyslogd(14737)─┬{in:imklog}(14779)
     │         │         │          │                      │
├{in:imuxsock}(14778)
     │         │         │          │                      │
└{rs:main Q:Reg}(14780)
     │         │         │          │                      ├sshd(15740)
     │         │         │          │                      ├systemd-
journal(13917)
     │         │         │          │                      └systemd-
logind(14970)
     │         │         │          ├mission-control(1868)─┬{dconf worker}(1872)
     │         │         │          │                      └{gdbus}(1870)
...

# pstree --highlight-pid=13151 --show-pids 13151  << only show a subset of the
process tree; from PID 13151 onward >>
systemd(13151)─┬─accounts-daemon(15045)─┬─{gdbus}(15274)
               │                        └─{gmain}(15079)
               ├─agetty(15993)
               ├─agetty(16013)
               ├─agetty(16033)
               ├─agetty(16053)
               ├─agetty(16073)
               ├─atd(15082)
               ├─cron(14669)
               ├─dbus-daemon(15253)
               ├─dhclient(14540)
               ├─irqbalance(15044)
               ├─polkitd(15400)─┬─{gdbus}(15437)
               │                └─{gmain}(15440)
               ├─rsyslogd(14737)─┬─{in:imklog}(14779)
               │                 ├─{in:imuxsock}(14778)
               │                 └─{rs:main Q:Reg}(14780)
               ├─sshd(15740)
               ├─systemd-journal(13917)
               └─systemd-logind(14970)
#
```

>>

Note:

- *[Lxd](#) ("lex-dee") from Canonical is a powerful container solution built on top of lxc ("lex-see").*

- From Brendan Gregg's blog:

"... ***Container Performance Analysis*** (DockerCon, 2017)

At DockerCon 2017 in Austin, I gave a talk on Linux container performance analysis, showing how to find bottlenecks in the host vs the container, how to profiler container apps, and dig deeper into the kernel.

A video of the talk is on youtube and the slides are on slideshare. ...”

<< P.T.O. >>

Namespace Isolation

Source

...

What's really interesting with Linux' approach to containers is precisely that it does *not* provide a "black-box/magical" container solution but instead provides individual isolation building blocks called "Namespaces", new ones appearing from releases to release. It also allows you to use solely the one you actually need for your specific application.

As of 3.12, Linux supports 6 Namespaces: *<< list below is re-ordered to reflect the order in which they were implemented on the Linux OS >>*

1. NS: mount points, first to land in Linux
2. UTS: hostname
3. IPC: inter-process communication
4. PID: "chroot" process tree
5. NET: network access, including interfaces
6. USER: map virtual, local user-ids to real local ones

... *[Source]*

The purpose of each namespace is to wrap a particular global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource.

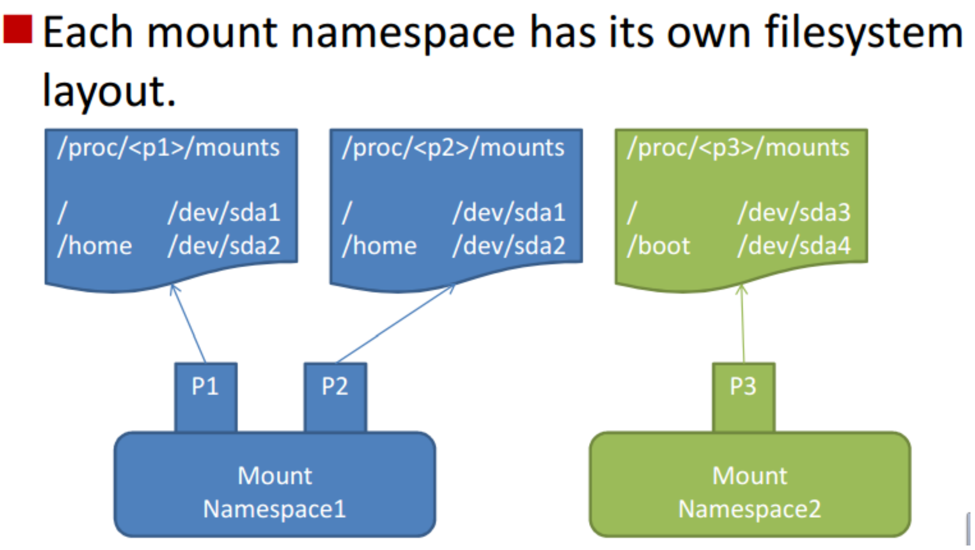
One of the overall goals of namespaces is to support the implementation of containers, a tool for lightweight virtualization (as well as other purposes) that provides a group of processes with the illusion that they are the only processes on the system.

...

From 'man 2 clone' :
*- can quickly lookup all **namespace isolation characteristics** that LXC exploits!*

1. Mount (NS) Namespace Isolation

```
CLONE_NEWNS (since Linux 2.4.19)
              If  CLONE_NEWNS  is set, the cloned child is started in a
new mount namespace, initialized with a copy of the namespace of the
parent.  If CLONE_NEWNS is not set, the child lives in the same mount
namespace as the parent. ...
```



[Source: above slide(s)]()

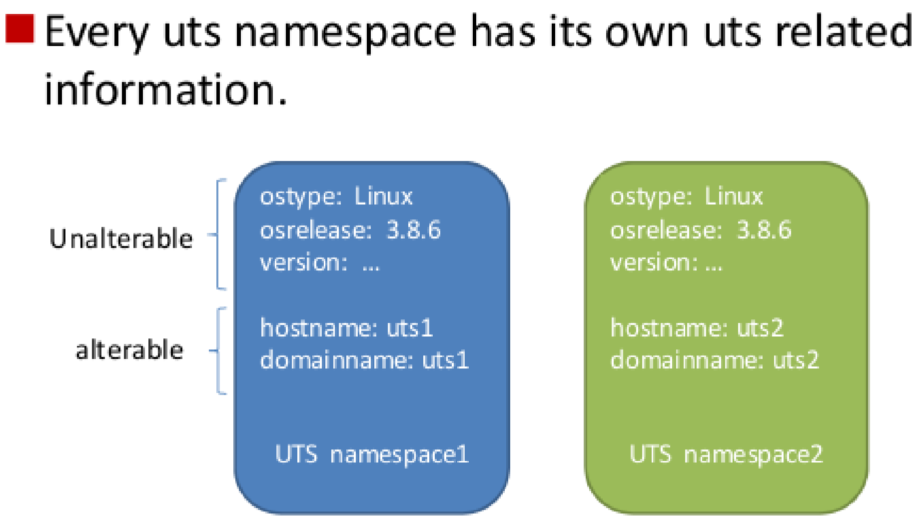
```
root@u1:/# ps -A
  PID TTY          TIME CMD
    1 ?        00:00:00 systemd
  373 ?        00:00:00 systemd-journal
  721 ?        00:00:00 dhclient
  821 ?        00:00:00 cron
  868 ?        00:00:00 rsyslogd
  988 ?        00:00:00 systemd-logind
 1026 ?        00:00:00 irqbalance
 1027 ?        00:00:00 accounts-daemon
 1046 ?        00:00:00 atd
 1139 ?        00:00:00 dbus-daemon
 1219 ?        00:00:00 polkitd
 1407 ?        00:00:00 sshd
 1547 lxc/console 00:00:00 agetty
 1557 pts/1    00:00:00 agetty
 1567 pts/2    00:00:00 agetty
 1577 pts/3    00:00:00 agetty
 1587 pts/0    00:00:00 agetty
 1810 ?        00:00:00 bash
 1818 ?        00:00:00 ps
```

```
root@u1:/# ls /proc/    << Namespace-isolated procfs for the container: here, proc only contains the data relevant to the processes alive on this container! >>
1     1547  373     buddyinfo  devices     fs        keys      mdstat    pagetypeinfo
softirqs      timer_list     vmstat
1026  1557  721     bus        diskstats    interrupts     key-users   meminfo
partitions    stat         timer_stats     zoneinfo
1027  1567  821     cgroups    dma         iomem     kmsg      misc      sched_debug
swaps    tty
1046  1577  868     cmdline    driver      ioports   kpagecount  modules   schedstat
sys           uptime
1139  1587  988     consoles   execdomains  irq           kpageflags  mounts    scsi
sysrq-trigger  version
1219  1810  acpi    cpuinfo    fb          kallsyms loadavg     mtrr      self     sysvipc
        version_signature
1407  1815  asound  crypto     filesystems  kcore    locks     net       slabinfo
thread-self     vmallocinfo
root@u1:/#
```

2. UTS Namespace Isolation

CLONE_NEWUTS (since Linux 2.6.19)

If CLONE_NEWUTS is set, then create the process in a new UTS namespace, whose identifiers are initialized by duplicating the identifiers from the UTS namespace of the calling process. If this flag is not set, then (as with fork(2)) the process is created in the same UTS namespace as the calling process. This flag is intended for the implementation of containers.



A UTS namespace is the set of identifiers returned by uname(2); among these, the domain name and the hostname can be modified by setdomainname(2) and sethostname(2), respectively. Changes made to the identifiers in a UTS namespace are visible to all other processes in the same namespace, but are not visible to processes in other UTS namespaces.

[Source: above slide(s)]

Only a privileged process (CAP_SYS_ADMIN) can employ CLONE_NEWUTS.

For further information on UTS namespaces, see namespaces(7).

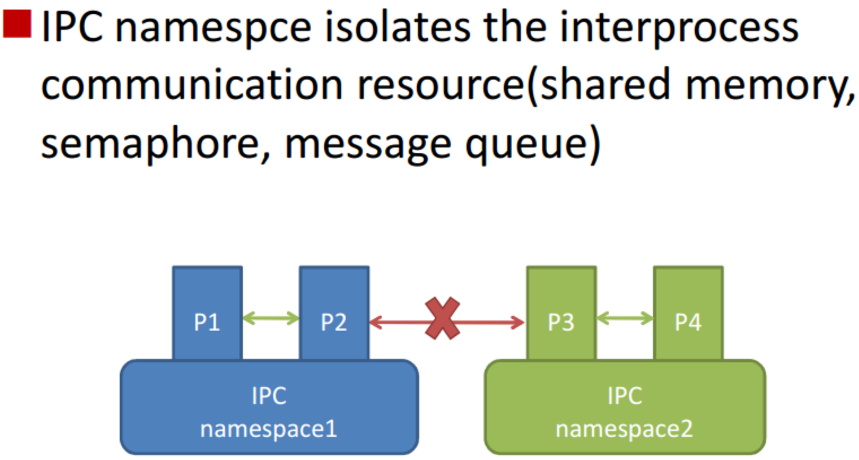
On the container

```
root@u1:/# uname -a
Linux u1 3.19.0-22-generic #22-Ubuntu SMP Tue Jun 16 17:15:15 UTC 2015
x86_64 x86_64 x86_64 GNU/Linux
root@u1:/#
```

<< For more details on UTS namespaces, and a very helpful example 'C' program, [see this link](). >>

3. IPC Namespace Isolation

```
...

CLONE_NEWIPC (since Linux 2.6.19)
              If CLONE_NEWIPC is set, then create the process in a new
IPC namespace. … An IPC namespace provides an isolated view of System V
IPC  objects  (see  svipc(7))  and (since Linux 2.6.30) POSIX message
queues (see mq_overview(7)). ...
```



[Source: above slide(s)]()

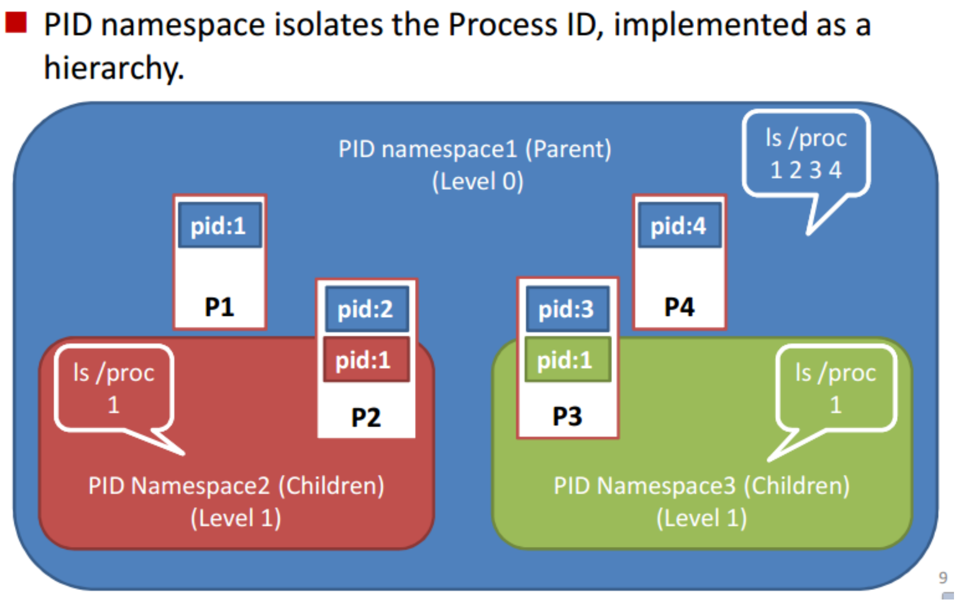
<< Below: Private SysV IPC namespace for the LXC container we created; here, no SysV IPC objects exist within this container (though some do exist on the host) >>

```
root@u1:/# ipcs -s

------ Semaphore Arrays --------
key        semid      owner      perms      nsems

root@u1:/# ipcs -q

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

root@u1:/# ipcs -m
```

```
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status

root@u1:/#
```

4. PID Namespace Isolation

```
CLONE_NEWPID (since Linux 2.6.24)
              If CLONE_NEWPID is set, then create the process in a new
PID namespace.  If this  flag  is not  set,  then  (as with fork(2)) the
process is created in the same PID namespace as the calling process.
This flag is intended for the implementation of containers.
...
```



Source: above slide(s)

<< The ps output shown earlier in this document clearly shows that PID's are in a isolated namespace from the host.. >>

Within the container:

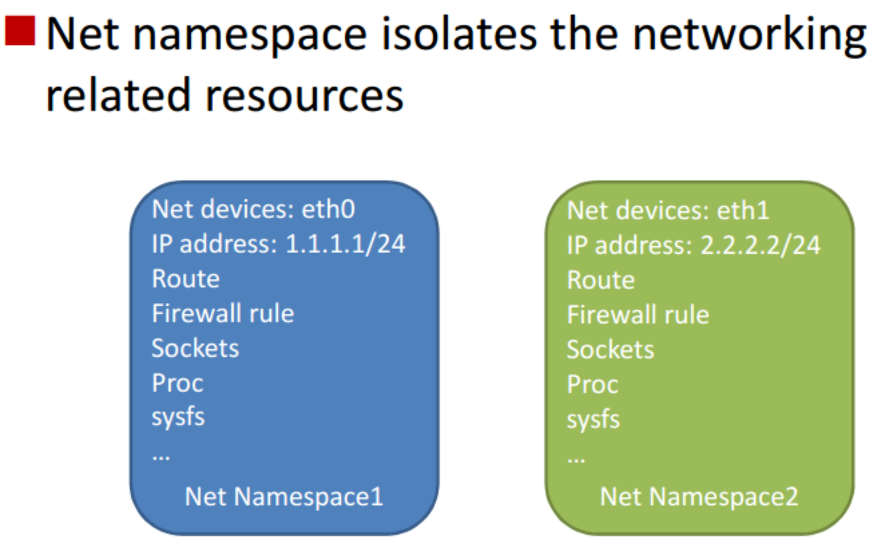
```
root@u1:/# ps -A
  PID TTY          TIME CMD
    1 ?        00:00:00 systemd
...
root@u1:/# ps ax|wc -l
21                     << few processes live within this container's PID
namespace >>
```

On the host:

```
$ ps ax|wc -l
343
```

5. Network Namespace Isolation

```
CLONE_NEWNET (since Linux 2.6.24)
              (The implementation of this flag was completed only by
about kernel version 2.6.29.) ...

             If CLONE_NEWNET is set, then create the process in a new
network namespace. … A network namespace provides an isolated view of
the  networking  stack  (network  device interfaces,  IPv4  and  IPv6
protocol  stacks,  IP  routing  tables,  firewall rules, the /proc/net
and /sys/class/net directory trees, sockets, etc.).  A physical  network
device can  live  in  exactly one network namespace.  A virtual network
device ("veth") pair provides a pipe-like abstraction that can be used
to create tunnels  between  network  namespaces,  and  can be used to
create a bridge to a physical network device in another namespace.
...
```



Within the container:

```
root@u1:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:16:3e:69:93:a5
          inet addr:10.0.3.87  Bcast:10.0.3.255  Mask:255.255.255.0
          inet6 addr: fe80::216:3eff:fe69:93a5/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:78 errors:0 dropped:0 overruns:0 frame:0
          TX packets:53 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:15893 (15.8 KB)  TX bytes:5623 (5.6 KB)
```

```
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@u1:/# ping -c1 bbc.co.uk
PING bbc.co.uk (212.58.244.20) 56(84) bytes of data.
64 bytes from fmt-vip71.telhc.bbc.co.uk (212.58.244.20): icmp_seq=1
ttl=46 time=144 ms

--- bbc.co.uk ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 144.277/144.277/144.277/0.000 ms
root@u1:/#
```

On the host system:

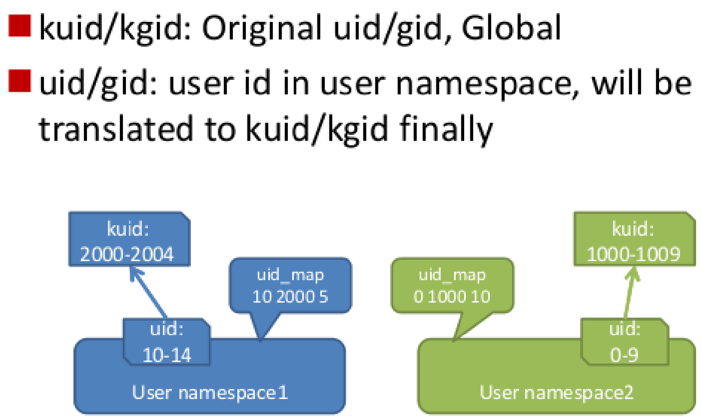
```
# ifconfig
...
vethQHQA61 Link encap:Ethernet  HWaddr fe:bb:ea:eb:3b:2b
          inet6 addr: fe80::fcbb:eaff:feeb:3b2b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:57 errors:0 dropped:0 overruns:0 frame:0
          TX packets:82 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:5918 (5.9 KB)  TX bytes:16371 (16.3 KB)
…
#
```

6. USER Namespace Isolation

CLONE_NEWUSER

(This flag first became meaningful for clone() in Linux 2.6.23, the current clone() semantics were merged in Linux 3.5, and the final pieces to make the user namespaces completely usable were merged in Linux 3.8.)

If CLONE_NEWUSER is set, then create the process in a new user namespace. If this flag is not set, then (as with fork(2)) the process is created in the same user namespace as the calling process. ...



Source: above slide(s)

… [Source]

User namespaces (CLONE_NEWUSER, started in Linux 2.6.23 and completed in Linux 3.8) isolate the user and group ID number spaces. In other words, a process's user and group IDs can be different inside and outside a user namespace. The most interesting case here is that a process can have a normal unprivileged user ID outside a user namespace while at the same time having a user ID of 0 inside the namespace. This means that the process has full root privileges for operations inside the user namespace, but is unprivileged for operations outside the namespace.

Starting in Linux 3.8, unprivileged processes can create user namespaces, which opens up a raft of interesting new possibilities for applications: since an otherwise unprivileged process can hold root privileges inside the user namespace, unprivileged applications now have access to functionality that was formerly limited to root. Eric Biederman has put a lot of effort into making the user namespaces implementation safe and correct. However, the changes wrought by this work are subtle and wide ranging. Thus, it may happen that user namespaces have some as-yet unknown security issues that remain to be found and fixed in the future.

…

More detailed LWN article:

…

One of the specific goals of user namespaces is to allow a process to have root privileges for operations inside the container, while at the same time being a normal unprivileged process on the wider system hosting the container.

To support this behavior, each of a process's user IDs has, in effect, two values: one inside the container and another outside the container. Similar remarks hold true for group IDs.

This duality is accomplished by maintaining a per-user-namespace mapping of user IDs: each user namespace has a table that maps user IDs on the host system to corresponding user IDs in the namespace.

This mapping is set and viewed by writing and reading the /proc/PID/uid_map pseudo-file, where PID is the process ID of one of the processes in the user namespace. Thus, for example, user

ID 1000 on the host system might be mapped to user ID 0 inside a namespace; a process with a user ID of 1000 would thus be a normal user on the host system, but would have root privileges inside the namespace. If no mapping is provided for a particular user ID on the host system, then, within the namespace, the user ID is mapped to the value provided in the file/proc/sys/kernel/overflowuid (the default value in this file is 65534). Our earlier article went into more details of the implementation.

...

Furthermore, see man pages for namespaces(7) and user_namespaces(7).

Recent:

```
$ man 2 clone
...
  CLONE_NEWCGROUP (since Linux 4.6)
              Create  the  process  in a new cgroup namespace.  If this flag is
not set, then (as with fork(2)) the process is created in the same cgroup
namespaces as  the  calling  process. This flag is intended for the
implementation of containers.

              For further information on cgroup namespaces, see
cgroup_namespaces(7). Only a privileged process (CAP_SYS_ADMIN) can employ
CLONE_NEWCGROUP.
...
$
```

LWN Resources:

Introduction to Linux namespaces – Part 5: NET

Jan 19, 2014 #linux #namespace

Introduction to Linux namespaces - Part 4: NS (FS)

Jan 12, 2014 #linux #namespace

Introduction to Linux namespaces - Part 3: PID

Jan 5, 2014 #linux #namespace

Introduction to Linux namespaces - Part 2: IPC

Dec 28, 2013 #linux #namespace

Introduction to Linux namespaces - Part 1: UTS

Dec 22, 2013

Also:

1. ***lxc-checkconfig*** - check the current kernel for lxc support

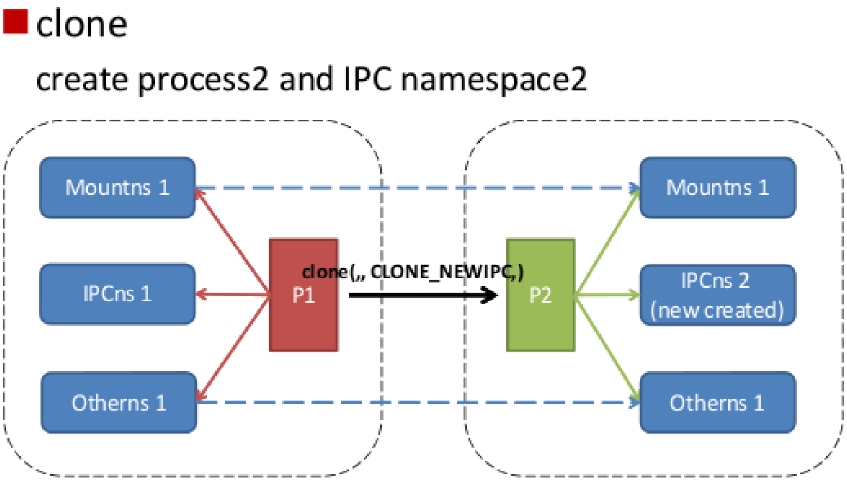
```
$ lxc-checkconfig
Kernel configuration not found at /proc/config.gz; searching...
Kernel configuration found at /boot/config-3.19.0-22-generic
--- Namespaces ---
Namespaces: enabled
Utsname namespace: enabled
Ipc namespace: enabled
Pid namespace: enabled
User namespace: enabled
Network namespace: enabled
Multiple /dev/pts instances: enabled

--- Control groups ---
Cgroup: enabled
Cgroup clone_children flag: enabled
Cgroup device: enabled
Cgroup sched: enabled
Cgroup cpu account: enabled
Cgroup memory controller: enabled
Cgroup cpuset: enabled

--- Misc ---
Veth pair device: enabled
Macvlan: enabled
Vlan: enabled
File capabilities: enabled

Note : Before booting a new kernel, you can check its configuration
usage : CONFIG=/path/to/config /usr/bin/lxc-checkconfig

$
```

2. "... the recent changes in the implementation of user namespaces are something of a game changer in terms of how namespaces can be used: starting with **Linux 3.8**, unprivileged processes can create user namespaces in which they have full privileges, which in turn allows any other type of namespace to be created inside a user namespace. ..." - *Namespaces in operation, part 1: namespaces overview, LWN, M Kerrisk*

System Calls that directly affect Namespaces

a. clone

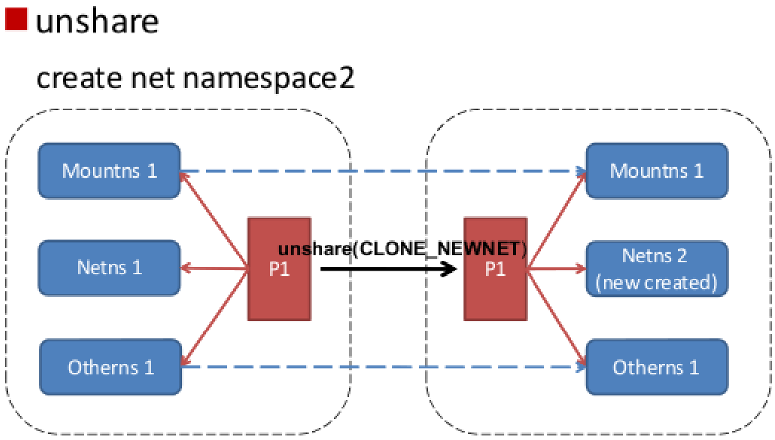
We've already discussed in some detail the clone(2) system call and the relevant clone flags:
Eg. of using *clone(... CLONE_NEWIPC ...);*



b. unshare

```
$ man 2 unshare
unshare - disassociate parts of the process execution context

SYNOPSIS
       #include <sched.h>

       int unshare(int flags);
...

DESCRIPTION
       unshare() allows a process to disassociate parts of its execution
context that are currently being shared with other processes.  Part of
the execution context, such as the mount namespace, is shared implicitly
when a new process is created using fork(2) or  vfork(2),  while  other
parts, such as virtual memory, may be shared by explicit request when
creating a process using clone(2).

       The main use of unshare() is to allow a process to control its
shared execution context without creating a new process.

       The  flags  argument  is  a  bit mask that specifies which parts
of the execution context should be unshared.  This argument is specified
by ORing together zero or more of the following constants:

       CLONE_FILES
              Reverse the effect of the clone(2) CLONE_FILES flag.
Unshare the file descriptor table, so that the calling process no longer
shares its file descriptors with any other process.

       CLONE_FS
              Reverse  the  effect  of the clone(2) CLONE_FS flag.
Unshare filesystem attributes, so that the calling process no longer
shares its root directory (chroot(2)), current directory (chdir(2)), or
umask (umask(2)) attributes with any other process.
```

```
       CLONE_NEWIPC (since Linux 2.6.19)
              This flag has the same effect as the clone(2) CLONE_NEWIPC
flag.  Unshare the System V IPC namespace, so that the calling process
has a  private copy of the System V IPC namespace which is not shared
with any other process.  Specifying this flag automatically implies
CLONE_SYSVSEM as well.  Use of CLONE_NEWIPC requires the CAP_SYS_ADMIN
capability.

       CLONE_NEWNET (since
Linux 2.6.24)
              This flag has
the same effect as the
clone(2) CLONE_NEWNET flag.
Unshare the network namespace,
so that the calling process is
moved into  a  new  network
namespace  which  is  not
shared  with  any  previously
existing  process.  Use of
CLONE_NEWNET requires the
CAP_SYS_ADMIN capability.
```



--snip--

An example 'C' program can be found in the [man page for unshare(2)].

Also, there is a CLI utility to achieve the same from within, say, a shell script: unshare(1). See the [man page for usage and several examples].

c. setns

$ man 2 setns

```
NAME
       setns - reassociate thread with a namespace

SYNOPSIS
       #define _GNU_SOURCE             /* See feature_test_macros(7) */
       #include <sched.h>

       int setns(int fd, int nstype);

DESCRIPTION
       Given a file descriptor referring to a namespace, reassociate the
calling thread with that namespace.
...
```

An example 'C' program can be found in the man page for setns(2).

Resources

Linux Containers and the Future Cloud , Rami Rosen, LJ, June 2014

Containers - not Virtuals Machines - are the Future Cloud, David Strauss, LJ, Jun 2013

Ubuntu Documentation: LXC

Tutorial: Getting Started with LXC on an Ubuntu 13.04 VPS

Isolation with Linux Containers

Namespaces in operation, part 1: namespaces overview, LWN, M Kerrisk

Thoughts on Linux Containers (LXC), Stefan Hajnoczi

(also has some information on Cgroups usage with cgmanager)

How does LXC virtualization work (under the hood)? Quora

Docker on Wikipedia

What does Docker add to just plain LXC?