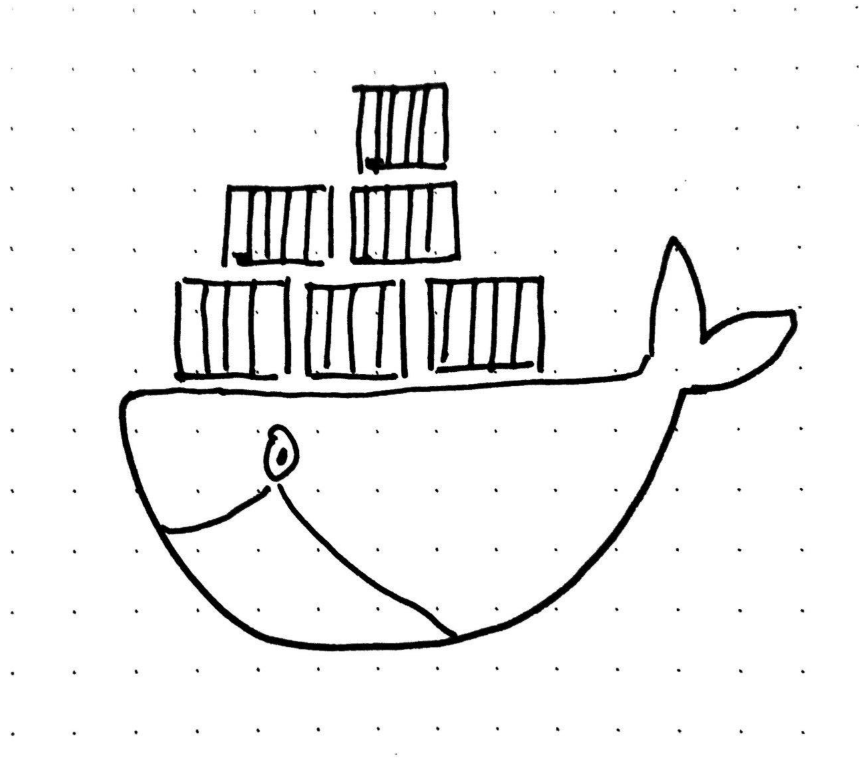
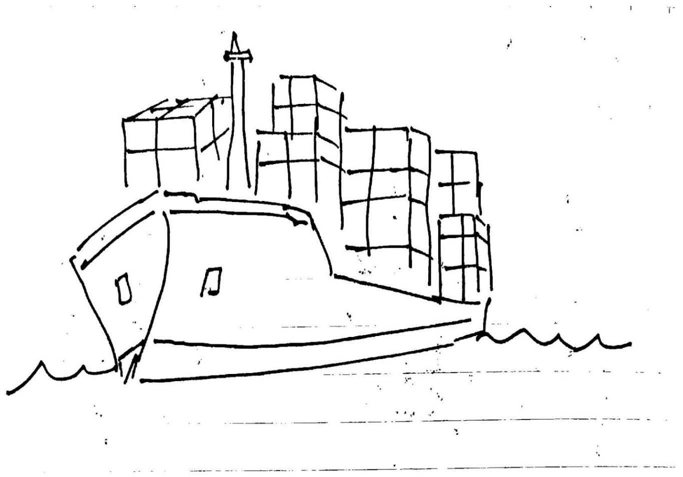
Namespaces.go

2018

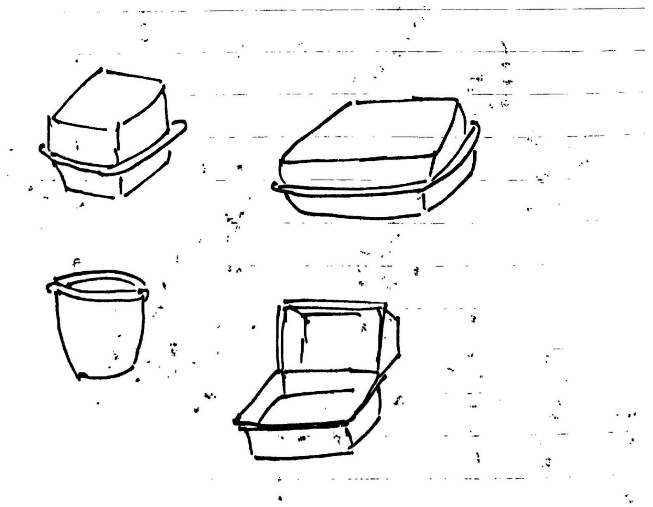


What is a container?

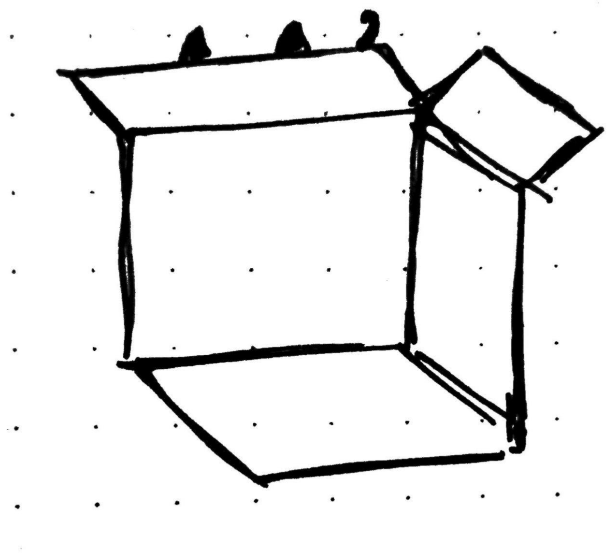
What is a container?



What is a container?



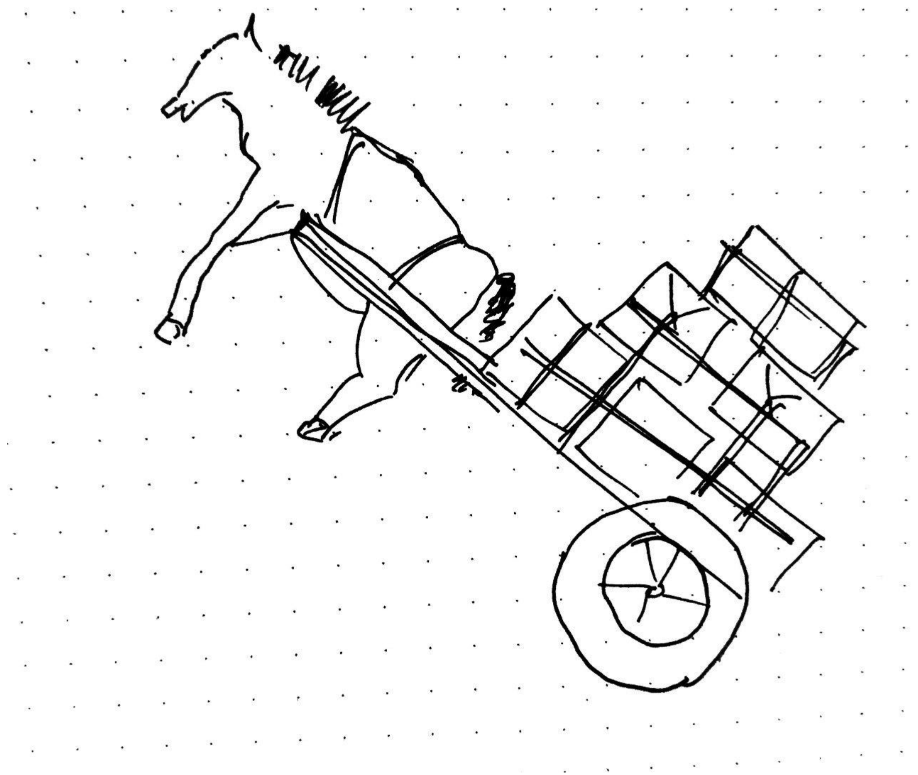
What is a container?



What is a container?

```
$ ps ax
  PID TTY      STAT   TIME COMMAND
    1 ?        Ss     4:19 /sbin/init splash
    2 ?        S      0:00 [kthreadd]
    3 ?        I<     0:00 [rcu_gp]
    4 ?        I<     0:00 [rcu_par_gp]
    6 ?        I<     0:00 [kworker/0:0H-kblockd]
    8 ?        I<     0:00 [mm_percpu_wq]
    9 ?        S      0:25 [ksoftirqd/0]
   10 ?        I      3:54 [rcu_sched]
   11 ?        I      0:00 [rcu_bh]
   12 ?        S      0:00 [migration/0]
```

Is container just a feeling? 💕



Abused containers?

Rewind: 2002-2006

name

namespace

What is a namespace?

- Partitions created by Linux kernel
- Isolate views of resources for Processes.
- Every Process has a namespace.
- Init starts with a default namespace.
- Two processes under same namespace have same access to resources.

Where are they?

```
$ ls -al /proc/10504/ns/

lrwxrwxrwx 1 cgroup -> cgroup:[4026531835]
lrwxrwxrwx 1 ipc -> ipc:[4026531839]
lrwxrwxrwx 1 mnt -> mnt:[4026531840]
lrwxrwxrwx 1 net -> net:[4026532009]
lrwxrwxrwx 1 pid -> pid:[4026531836]
lrwxrwxrwx 1 user -> user:[4026531837]
lrwxrwxrwx 1 uts -> uts:[4026531838]
```

```
$ ls -al /proc/3330/ns/

lrwxrwxrwx 1 cgroup -> cgroup:[4026531835]
lrwxrwxrwx 1 ipc -> ipc:[4026531839]
lrwxrwxrwx 1 mnt -> mnt:[4026531840]
lrwxrwxrwx 1 net -> net:[4026532009]
lrwxrwxrwx 1 pid -> pid:[4026531836]
lrwxrwxrwx 1 user -> user:[4026531837]
lrwxrwxrwx 1 uts -> uts:[4026531838]
```

Types of namespaces

- Mount
- UTS
- IPC
- PID
- Network
- User
- Cgroup

Containers?

- Containers are to Process what Process are to Threads.
- Virtualization
- Less sharing
- More Separation

Sharing is not Caring,
Your parents have been lying.

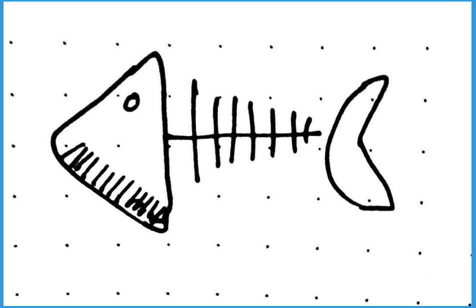
Syscall?

```go
func main() {
    pid, _, _ := syscall.Syscall(syscall.SYS_GETPID, 0, 0, 0)
    fmt.Println("process id: ", pid)
}
```

Syscall

```
$ go run hello.go
process id:  12566
```

Skeleton



framework

```
func main() {
    cmd := exec.Command("/bin/sh")
    cmd.Run()
}

$ go run hello.go
$
```

framework

```
func main() {
    cmd := exec.Command("/bin/bash")

    cmd.Stdin = os.Stdin
    cmd.Stdout = os.Stdout
    cmd.Stderr = os.Stderr

    cmd.Run()
}
```

framework

```
$ sudo ./main
root@xps:~/workspace/meson10/test# exit
```

Helper functions

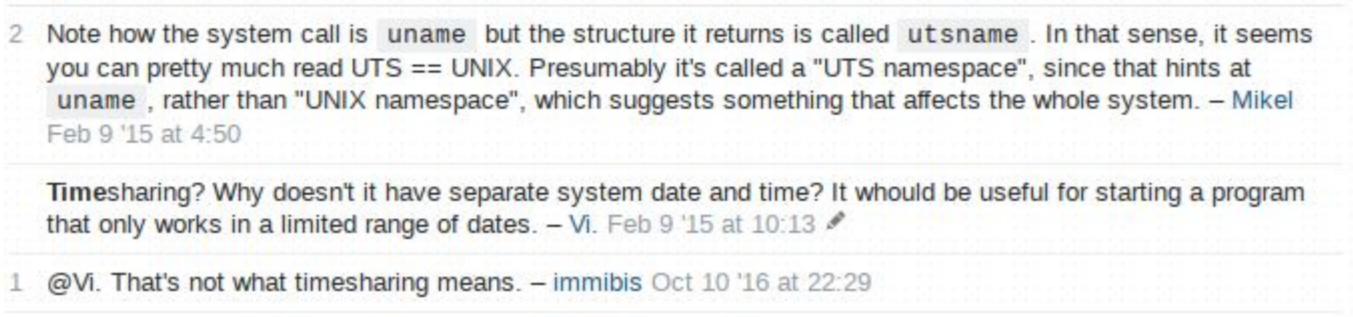
```go
// Attaches stdin, stdout, stderr to Cmd.
func makeCmd(cmd *exec.Cmd) *exec.Cmd {
    cmd.Stdin = os.Stdin
    cmd.Stdout = os.Stdout
    cmd.Stderr = os.Stderr
    return cmd
}
```

UTS namespace

UTS namespace

- Isolate uname system call.
- hostname
- domainname

UTS namespace



2 Note how the system call is `uname` but the structure it returns is called `utsname`. In that sense, it seems you can pretty much read UTS == UNIX. Presumably it's called a "UTS namespace", since that hints at `uname`, rather than "UNIX namespace", which suggests something that affects the whole system. – Mikel Feb 9 '15 at 4:50

Timesharing? Why doesn't it have separate system date and time? It whould be useful for starting a program that only works in a limited range of dates. – Vi. Feb 9 '15 at 10:13

1 @Vi. That's not what timesharing means. – immibis Oct 10 '16 at 22:29

UTS namespace

```
func main() {
    cmd := makeCmd(exec.Command("/bin/sh"))
    cmd.SysProcAttr = &syscall.SysProcAttr{
        Cloneflags: syscall.CLONE_NEWUTS,
    }
    cmd.Run()
}
```

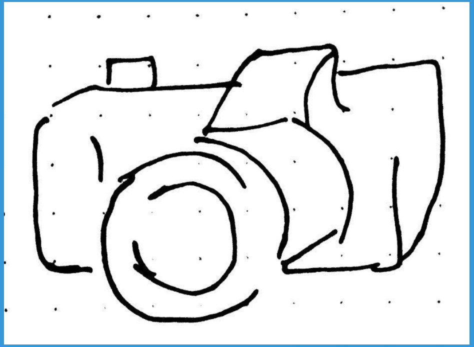
UTS namespace

```
$ sudo ./main
# hostname hello
# hostname
Hello
```

ANOTHER SHELL

```
$ hostname
xps.piyushverma.net
```

Picture or it didn't happen



$: sudo go run hello.go

```
20078 pts/1    S      0:00   |   |   |   \_ sudo ./hello
20079 pts/1    Sl     0:00   |   |   |       \_ ./hello
20084 pts/1    S+     0:00   |   |   |           \_ /bin/bash
```

User namespace

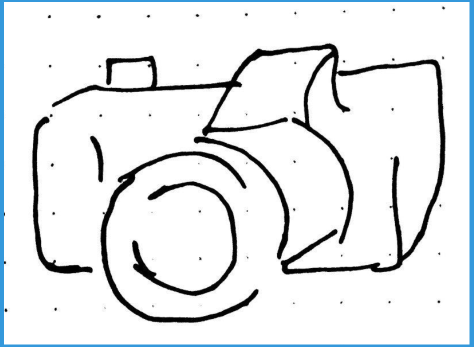
User namespace

```go
func main() {
    cmd := makeCmd(exec.Command("/bin/sh"))
    cmd.Env = []string{"PS1=[gophercon]$ "}

    cmd.SysProcAttr = &syscall.SysProcAttr{
        Cloneflags: syscall.CLONE_NEWUSER,
    }
    cmd.Run()
}
```

UID/GID

```
$ echo $UID
1000

$ go run user_ns.go
[gophercon]$ echo $UID
[gophercon]$
```

Picture or it didn’t happen



User namespace

```
$ ls -al /proc/5054/ns/

lrwxrwxrwx cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx ipc -> 'ipc:[4026531839]'
lrwxrwxrwx mnt -> 'mnt:[4026532471]'
lrwxrwxrwx net -> 'net:[4026532008]'
lrwxrwxrwx pid -> 'pid:[4026531836]'
lrwxrwxrwx user -> 'user:[4026531837]'
lrwxrwxrwx uts -> 'uts:[4026531838]'
```

```
$ ls -l /proc/3692/ns/

lrwxrwxrwx cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx ipc -> 'ipc:[4026531839]'
lrwxrwxrwx mnt -> 'mnt:[4026532471]'
lrwxrwxrwx net -> 'net:[4026532008]'
lrwxrwxrwx pid -> 'pid:[4026531836]'
lrwxrwxrwx user -> 'user:[4026532334]'
lrwxrwxrwx uts -> 'uts:[4026531838]'
```

User namespace

```
$ go run user_ns.go
```

```
[gophercon]$ whoami
nobody
```

UID/GID

```
cmd := makeCmd(exec.Command("/bin/sh"))
cmd.SysProcAttr = &syscall.SysProcAttr{
	Cloneflags: syscall.CLONE_NEWUSER,
	UidMappings: []syscall.SysProcIDMap{{
		ContainerID: 109,
		HostID:      os.Getuid(),
		Size:        1,
	}},
	GidMappings: []syscall.SysProcIDMap{{
		ContainerID: 114,
		HostID:      os.Getgid(),
		Size:        1,
	}},
}
```

UID/GID

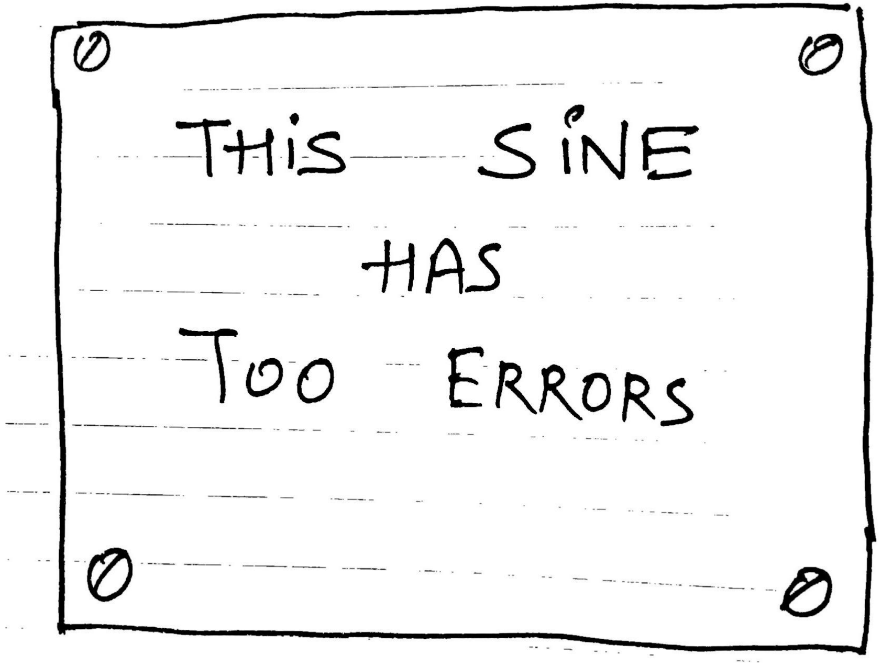
```
$ go run user_ns.go

[gophercon]$ whoami
grafana

[gophercon]$ groups
gdm
```

Real Problem

Problem

```
func main() {
    cmd := makeCmd(exec.Command("/bin/sh "))
    cmd.SysProcAttr = &syscall.SysProcAttr{
        Cloneflags: syscall.CLONE_NEWUTS,
    }

    syscall.Sethostname([]byte("inner-system"))
    cmd.Run()
}
```

/proc/self/exe



/proc/self/exe

```
$ ls -al /proc/self/exe

lrwxrwxrwx 1 Dec  7 19:41 /proc/self/exe -> /bin/ls
```

/proc/self/exe

```
$ cat /bin/cat | wc
     154     767   43256

$ cat /proc/self/exe | wc
     154     767   43256
```

Helper functions

Helper.go

```
func inContainer() bool {
    i := flag.Bool("inner", false, "child")
    flag.Parse()
    return *i
}
```

```
func main() {
    if inContainer() {
        inner()
    } else {
        run()
    }
}
```

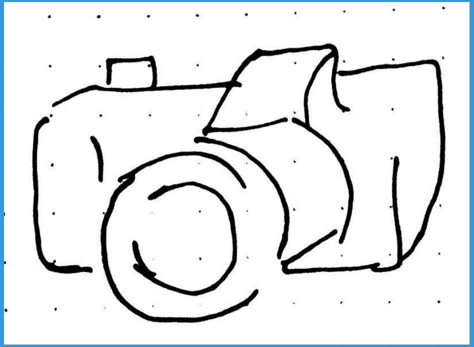
```go
func run() error {
    cmd := makeCmd(exec.Command("/proc/self/exe", "-inner"))
    cmd.SysProcAttr = &syscall.SysProcAttr{
        Cloneflags: syscall.CLONE_NEWUTS,
    }
    return cmd.Run()
}

func inner() error {
    syscall.Sethostname([]byte("inner-system"))
    return makeCmd(exec.Command("/bin/sh")).Run()
}
```

UTS namespace

```
$ go build uts.go
$ sudo ./uts
```

```
[gophercon]$ hostname
inner-system
```

Picture or it didn’t happen



/proc/self/exe

```
19406 pts/1  S    0:00  |  |  |  \_ sudo ./hello
19416 pts/1  Sl   0:00  |  |  |      \_ ./hello
19421 pts/1  Sl   0:00  |  |  |          \_ /proc/self/exe -inner
19426 pts/1  S+   0:00  |  |  |              \_ /bin/sh
```

Reexec

https://github.com/moby/moby/tree/master/pkg/reexec

PID namespace

PID namespace

```
func run() error {
    cmd := makeCmd(exec.Command("/proc/self/exe", "-inner"))
    cmd.SysProcAttr = &syscall.SysProcAttr{
        Cloneflags: syscall.CLONE_NEWPID,
    }
    return cmd.Run()
}
func inner() error {
    fmt.Println("Inner code PID", os.Getpid())
}
```

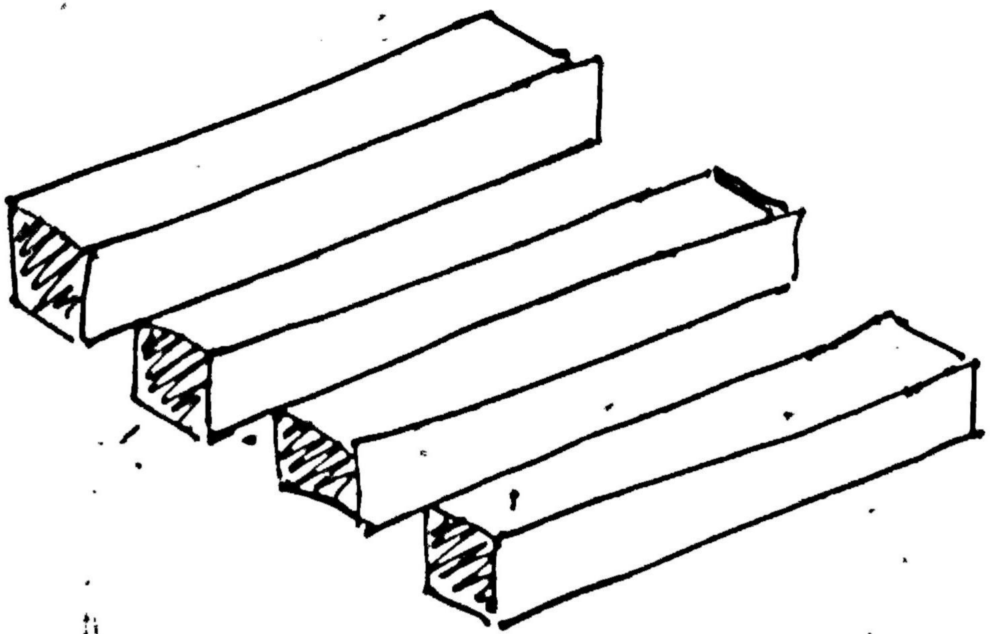
PID namespace

```
$ go build main.go

$ sudo ./main
Inner code PID 1
[gophercon]$
```

Problem

```
[gophercon]$ ps ax
  PID TTY      STAT   TIME COMMAND
    1 ?        Ss     0:03 /sbin/init splash
    2 ?        S      0:00 [kthreadd]
    4 ?        S<     0:00 [kworker/0:0H]
    6 ?        S<     0:00 [mm_percpu_wq]
    7 ?        S      0:01 [ksoftirqd/0]
    8 ?        S      1:32 [rcu_sched]
    9 ?        S      0:00 [rcu_bh]
   10 ?        S      0:00 [migration/0]
```

Mnt namespace



Perspective

```
$ sudo ./main
Inner code PID 1
[gophercon]$ findmnt -o+PROPAGATION
TARGET                                      OPTIONS
PROPAGATION
/                          rw,noatime,errors=remount-ro,data=ordered                          shared
|-/dev                     rw,nosuid,relatime,size=7622836k,nr_inodes=1905709,mode=755         shared
| |-/dev/pts               rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000               shared
| |-/dev/shm               rw,nosuid,nodev                                                     shared
| |-/dev/mqueue            rw,relatime                                                         shared
| `-/dev/hugepages         rw,relatime,pagesize=2M                                             shared
|-/run                     rw,nosuid,noexec,relatime,size=1530088k,mode=755                    shared
| |-/run/lock              rw,nosuid,nodev,noexec,relatime,size=5120k                          shared
| `-/run/user/1000         rw,nosuid,nodev,relatime,size=1530084k,mode=700,uid=1000,gid=1000   shared
|-/sys                     rw,nosuid,nodev,noexec,relatime                                     shared
| |-/sys/kernel/security   rw,nosuid,nodev,noexec,relatime                                     shared
```

mnt problem

Mnt namespace

```
func run() error {
    cmd := makeCmd(exec.Command("/proc/self/exe", "-inner"))
    cmd.SysProcAttr = &syscall.SysProcAttr{
        Cloneflags: syscall.CLONE_NEWNS
    }
    return cmd.Run()
}
func inner() error {
    return makeCmd(exec.Command("/bin/sh")).Run()
}
```

```
$ sudo ./main
Inner code PID 1
[gophercon]$ findmnt -o+PROPAGATION
TARGET                                  OPTIONS
PROPAGATION
/                          rw,noatime,errors=remount-ro,data=ordered                   private
|-/dev                     rw,nosuid,relatime,size=7622836k,nr_inodes=1905709,mode=755 private
| |-/dev/pts               rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000       private
| |-/dev/shm               rw,nosuid,nodev                                             private
| |-/dev/mqueue            rw,relatime                                                 private
| `-/dev/hugepages         rw,relatime,pagesize=2M                                     private
|-/run                     rw,nosuid,noexec,relatime,size=1530088k,mode=755            private
| |-/run/lock              rw,nosuid,nodev,noexec,relatime,size=5120k                  private
| `-/run/user/1000         rw,nosuid,nodev,relatime,size=1530084k,mode=700,uid=1000,gid=1000 private
|-/sys                     rw,nosuid,nodev,noexec,relatime                             private
| |-/sys/kernel/security   rw,nosuid,nodev,noexec,relatime                             private
```

mnt problem

Unshare

unshare

allows a process (or thread) to disassociate parts of its execution context that are currently being shared with other processes (or threads). 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 or thread using clone(2).

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

unshare flags

```
func run() error {
    cmd := makeCmd(exec.Command("/proc/self/exe", "-inner"))
    cmd.SysProcAttr = &syscall.SysProcAttr{
        Cloneflags: syscall.CLONE_NEWNS | syscall.CLONE_NEWPID,
        Unshareflags: syscall.CLONE_NEWNS,
    }
    return cmd.Run()
}
func inner() error {
    syscall.Mount("/proc", "/proc", "proc", uintptr(0), "")
```

PID (isolated)

```
$ go build main.go
$ sudo ./main

[gophercon]$ ps -aexf
  PID TTY      STAT   TIME COMMAND
    1 pts/0    Sl     0:00 /proc/self/exe -inner PS1=[gophercon]$
    6 pts/0    S      0:00 /bin/sh PS1=[meson10]$
    7 pts/0    R+     0:00  \_ ps -aexf PS1=[gophercon]$  PWD=/
```

Added in go 1.9

https://github.com/golang/go/issues/19661

"It turns out that the systemd developers decided to override the kernel's default setting of 'private' to their own default setting of 'shared'. This means that on Linux machines with systemd, the default is shared , while on Linux machines without systemd, the default is private. Essentially, systemd decided to make it so that there is no default that end programs can rely on. All programs must instead mark the root filesystem as private if they want private namespaces, or as shared if they want shared namespaces if they want to work across all Linux distributions. **I'm pretty sure this was done to frustrate as many people as possible.**"

Homework

- IPC
- Net
- CGroup

xps:~$ whoami

Piyush Verma
Site Reliability Engineering
Trusting Social

Twitter: **meson10**

Thank you.

Credits

- Sagar Rakshe
- Mohan Dutt Parashar
- Talina Shrotriya
- Akshat Goyal