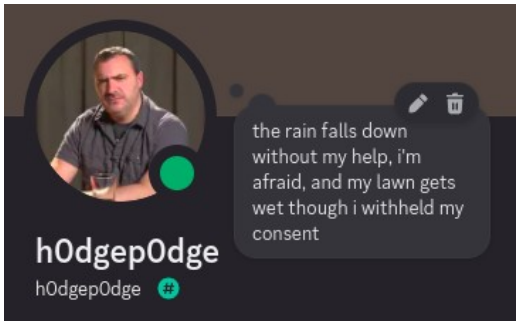
What’s in a name(space)?

Robbie Mckennie

Who am I?

Robbie
Isolation, containerization, abstraction



What is this talk for?

- A tour of Linux Namespaces
- Not a lecture
- Check out the linux manual it’s sick

what is this talk?
- a fast paced guided tour, sharing my learning exploring this world, not a lecture
- give you an overview so when you encounter a weird situation, you can remember "hey i remember namespaces, that might help here"
- inspire you to read obscure man pages because they're surprisingly interesting and cool
    - nftables, iptables, traffic control, pppd, socat (see how they're all network stuff)

What is a namespaces?

Noun [edit]

namespace (*plural* **namespaces**)

1. (*programming*) A conceptual space that groups classes, identifiers, etc. in order to avoid conflicts with items in unrelated code that carry the same names.

- A scope within which the same name refers to the same object, but outside of that scope that name may refer to something different.
- In the context of the Linux kernel, a namespace virtualizes and isolates the resources on the system
- The funamental foundation for Linux containers
- A process has a particular set of namespaces it operates within, which are typically inherited from the parent process
- Automatically garbage collected

```
rob@orobas:~$ ls -l /proc/self/ns
total 0
lrwxrwxrwx 1 rob rob 0 Mar 10 20:32 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 rob rob 0 Mar 10 20:32 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 rob rob 0 Mar 10 20:32 mnt -> 'mnt:[4026531841]'
lrwxrwxrwx 1 rob rob 0 Mar 10 20:32 net -> 'net:[4026531840]'
lrwxrwxrwx 1 rob rob 0 Mar 10 20:32 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 rob rob 0 Mar 10 20:32 pid_for_children -> 'pid:[4026531836]'
lrwxrwxrwx 1 rob rob 0 Mar 10 20:32 time -> 'time:[4026531834]'
lrwxrwxrwx 1 rob rob 0 Mar 10 20:32 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 rob rob 0 Mar 10 20:32 user -> 'user:[4026531837]'
lrwxrwxrwx 1 rob rob 0 Mar 10 20:32 uts -> 'uts:[4026531838]'
rob@orobas:~$
```

what is a namespaces? - broadly, a namespace is a scope where the same name refers to the same object, and outside of that scope that name may refer to a different object - in the context of linux, namespaces virtualize and isolate resources on the system, changing the view that a process has of the system - can be hierarchical or non-hierarchical - automatically inherited by child processes - joined by reference to a process, file descriptor, or bind mounted file - creating a namespace puts the current process into that namespace, or puts all child processes into that namespace - garbage collected when all references are lost

Your toolkit

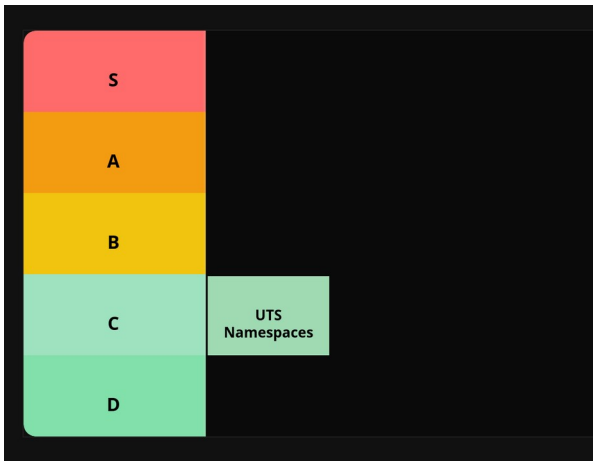
- unshare
- nsenter
- lsns
- /proc/*/ns
- ip netns
- RTFM: namespaces(7), *_namespaces(7)

UTS Namespaces

- Isolates the system hostname and domain name
- What's UTS? Unix time system, some historic mumbo jumbo

```
struct utsname {
    char sysname[];    /* Operating system name (e.g., "Linux") */
    char nodename[];   /* Name within communications network
                          to which the node is attached, if any */
    char release[];    /* Operating system release
                          (e.g., "2.6.28") */
    char version[];    /* Operating system version */
    char machine[];    /* Hardware type identifier */
#ifdef _GNU_SOURCE
    char domainname[]; /* NIS or YP domain name */
#endif
};
```

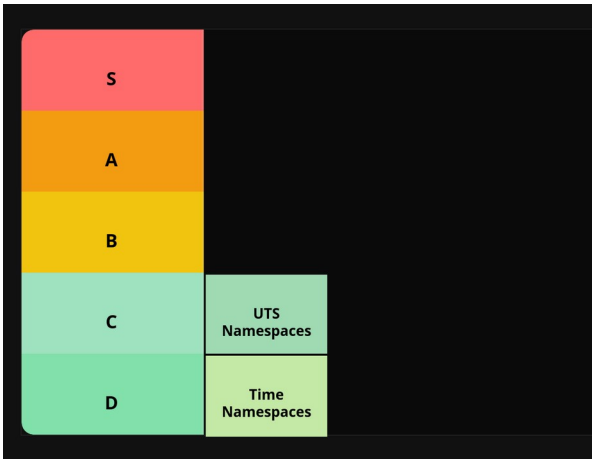
uts namespaces
- isolates the host name and host domain
- what even is uts? unix time system, historical mumbo jumbo
- demo: unshare --uts, hostname foobar, uname -a, bash
- boring but ultimately necessary, low tier


S
A
B
C
D
UTS
Namespaces

Time Namespaces

- Creates a virtual system clock, but only for CLOCK_MONOTONIC and CLOCK_BOOTTIME
- The real time clock and clocks derived therefrom are not included
- Something about monotonicity and undefined behaviour
- Virtual clocks are given a fixed offset from the global clock

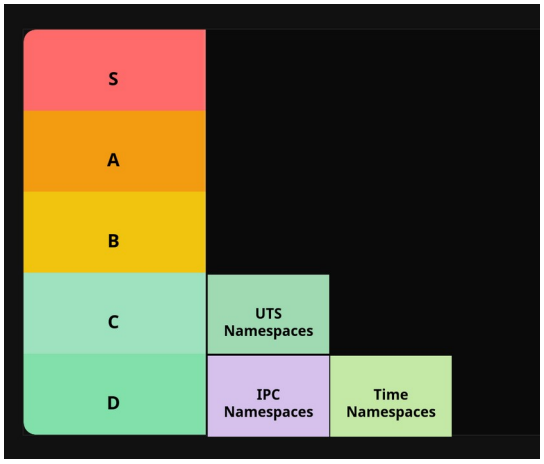
time namespaces
- virtualizes system clocks
- only child processes are placed in the new namespace
- adds fixed offsets relative to parent namespace
- "The motivation for adding time namespaces was to allow the monotonic and boot-time clocks to maintain consistent values during container migration and checkpoint/restore."
- I thought this would also involve giving difference processes different timezones, but apparently not
- Man page gives a demo which I believe is fundamentally broken
    - Offsets must be configured before the first child process is created
    - unshare(1) creates the new namespace for you, but then exec's a new process immediately
    - I got this demo working by writing a bash builtin that called unshare(2) without forking, but it's not really that useful
- demo using unshare -T --monotonic 5000 --boottime 5000 --bash
- low tier. A failed promise


S
A
B
C
D
UTS
Namespaces
Time
Namespaces

IPC Namespaces

- Inter-process communication, but not the type you're thinking of (not signals or unix domain sockets (these are found in PID and mount namespaces respectively))
- System V IPC and POSIX message queues
- IPC namespaces create isolated islands of processes that can communicate through these mechanisms

ipc
- "inter process communication": system V IPC sysvipc(7), POSIX message queues mq_overview(7)
  - sysV IPC: messages, semaphores, shared memory
  - POSIX message queues: messages, queues are identified by a human readable name
- IPC namespaces create isolated islands of processes that can communicate through these mechanisms
- low tier. what is this, the 80s?

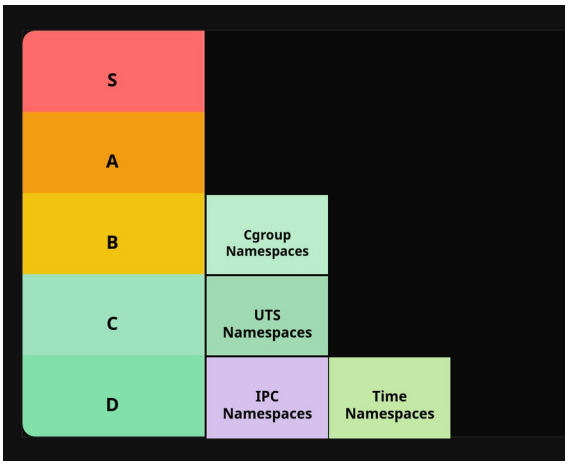

S
A
B
C
D
UTS Namespaces
IPC Namespaces
Time Namespaces

Cgroup Namespaces

- Cgroups??
  - Process control by organizing processes into hierarchical groups, pretty cool
  - Is a filesystem, managed by editing files and creating directories
  - Restrictions placed on a group apply to all child groups
- Cgroup namespaces change the view of processes in that namespace, to see a given cgroup as THE root cgroup
- Benefits include being able to run a process that assumes it is in the root cgroup somewhere other than the root cgroup, and being able to hide the rest of the cgroup hierarchy from a process you don't want to be able to access it

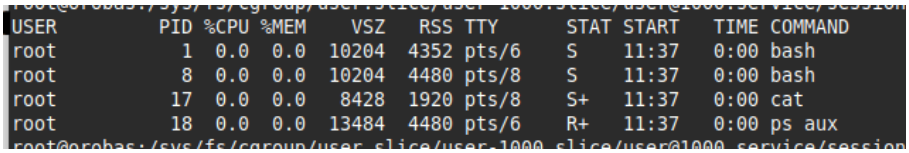
cgroup namespaces
- cgroups?!
  - hierarchical groups controlling and monitoring resource usage (cpu time, memory usage, disk i/o, etc)
    - that's right, monitoring, by group. you can see how much cpu and memory each _user_ is hogging, or monitor a mutli-process application.
    - surprised to learn you can freeze processes, when I know full well you can freeze processes and do it all the time (ctrl-z)
  - Managed using a filesystem?
    - Settings configured with files, and new groups are created by creating directories, very linuxy
    - I need a hierarchy, my filesystem is a hierarchy ¯_(ツ)_/¯ recommended viewing: https://www.youtube.com/watch?v=j10y76Px-cI
  - (mostly) each subsystem has a kernel component called a controller, in charge of the resources in that subsystem
  - a controller may be enabled in a particular group, but disabled for all child groups (subtree_control)
- cgroup namespaces shift the perspective of child processes so that they see their cgroup as the root cgroup
- allows processes to operate with no knowledge of the parent hierarchy
- prevents a process from accessing or modifying the controls from parent groups
- mid-high tier. cgroups are wicked, and their namespaces seem to have legitimate uses

https://www.youtube.com/watch?v=j10y76Px-cI


S
A
B
C
D
Cgroup Namespaces
UTS Namespaces
IPC Namespaces
Time Namespaces

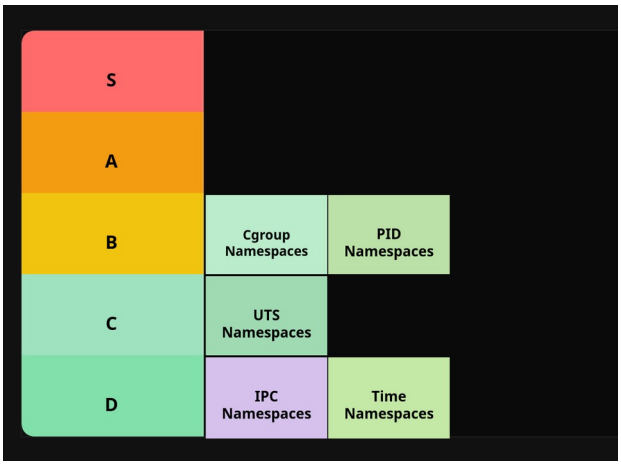
PID Namespaces

- Hides all processes outside the namespace from those inside
- Hierarchical, processes are allocated a PID in each of their parent namespaces, but can only interact with those in the same namespace or descendent namespaces.
- The kernel will handle translation of the PID between namespaces when needed



pid

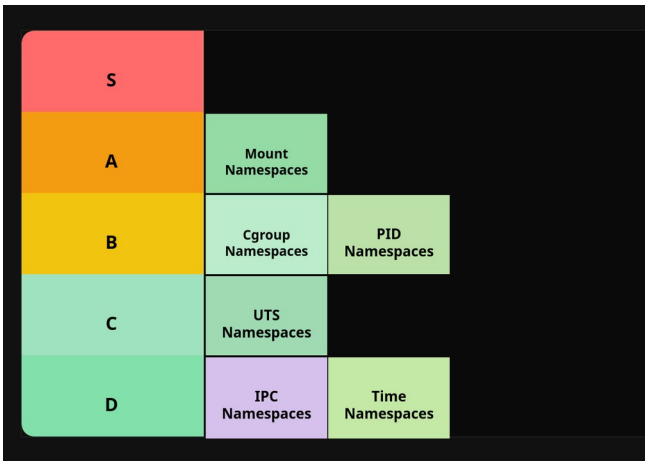
- pid namespaces hide all processes outside the namespace from the processes inside
- processes still have a pid in all parent namespaces, and their new pid in the new namespace is usually different
- when a pid is passed through a unix socket between processes in different namespaces, the pid is translated to the equivalent pid in the other namespace, pretty cool.
- demo with unshare –pid –fork bash
- a proc filesystem shows only processes in the pid namespace of the process that performs the mount
- mid tier. useful for containers, a mere curiosity for me


S
A
B
C
D
Cgroup Namespaces
PID Namespaces
UTS Namespaces
IPC Namespaces
Time Namespaces

Mount Namespaces

- Mounts, the attachment of filesystems to mount points
- At a basic level, mounts created within a namespace aren’t visible outside of the namespace
  - (Unless you specify that the mount should propogate between namespaces, but it gets pretty complicated)
- Mounts also have relationships with other namespaces, cgroup mounts are governed by cgroup namespaces, proc mounts are governed by PID namespaces
- This means we can now get ps working in a PID namespace!

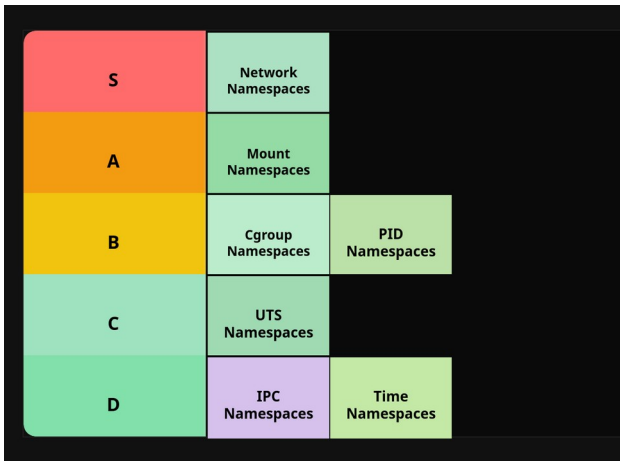
mount
- we all know what mounts are right?
- newly created mount namespaces inherit existing mounts from their parent namespace, but (by default) new mounts and unmounts in one namespace do not affect the other.
  - a mount may be designated as shared, and changes made to that mount and sub-tree mounts will propogate between namespaces that have that mount.
- there's a lot of detail in mount namespaces, but it gets very abstract
- pivot_root(2): namespace aware equivalent of chroot
- unshare –mount –pid –fork bash, mount -t proc proc /proc, ps aux
- high tier. very useful, but doesn't come up very much in what I do


S
A
B
C
D
Mount Namespaces
Cgroup Namespaces
PID Namespaces
UTS Namespaces
IPC Namespaces
Time Namespaces

Network Namespaces

- Virtualizes the networking stack, creating an independant interface list, routing tables, firewall rules, and network stack configuration.
- Once created, interfaces from another interface can be moved between them, including physical interfaces.
- Cool for isolating a process to using a particular interface, or keep an interface away from a process broadcasting noise down the line.
- “Virtual ethernet” interfaces can also be created to link network namespaces together.

network
- isolate devices, routing and bridging tables, firewall rules, traffic control
- i love them for running packet capture without a dhcp client or etc messing with my interface or poluting the capture
- you can bottle up an application to keep it from accessing the internet or the local network
- you can restrict a process to only being able to connect to a particular interface
- ip netns add left, ip netns add right, ip link add type veth, ip link set veth0 netns left, ip link set veth1 netns right, ip -n left addr add 192.168.0.1/24 dev veth0, ip -n left addr add 192.168.0.2/24 dev veth1, ip -n left link set veth0 up, ip -n right link set veth1 up
- high tier, i find namespaces useful on a regular basis

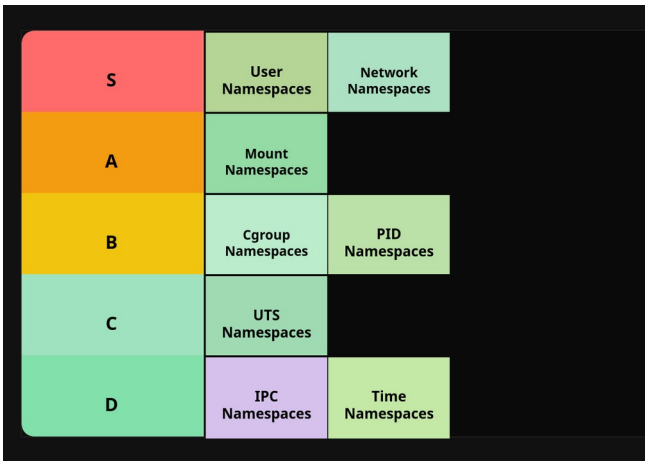

S
A
B
C
D
Network Namespaces
Mount Namespaces
Cgroup Namespaces
PID Namespaces
UTS Namespaces
IPC Namespaces
Time Namespaces

User Namespaces

- User namespaces create a virtualized structure of users and groups, provides uid and gid translation between namespaces, and virtualizes permissions relating to namespaces.
- When a user would normally need special permissions to do certain operations, now all that user needs is permission to do that operation _in the present user namespace_
- This allows you to have a root user in a container that has full admin permissions inside that container, but has no permissions to do anything outside of that container.
- When a new user namespace is created, the creating process gains full admin powers within that namespace, to create mounts (if the present mount namespace is owned by the present user namespace), to kill processes (if the present PID namespace is owned by the present user namespace), to manage cgroups (if the present cgroup namespace is owned by the present user namespace), and so on.
- And the kicker...

user

- user namespace is a bit of a misnomer, user namespaces isolate not just user and group ids (with mappings back to the ids in the parent namespace), but also provides virtualized permissions for child namespaces.
- uid/gid mapping, transforms between user namespaces. for the purpose of permission checks on files, the id is translated through the hierarchy back to the root namespace, and for the purpose of reading the ownership ids of a file, the translation is done in the opposite direction.
    - for example, if i create a user namespace and map my user id to 0 (the root UID), to processes within that namespace files owned by my user will appear to be owned by root, and any files created in that user namespace will be owned by my user in the root user namespace.
    - historical tidbit: prior to 4.15, the number of mappings was limited to 5. in 4.15 and after, that limit was extended to 340, what a random number
    - if you've had issues with docker containers messing with file ownership, this is the source of that problem
- when a new non-user namespace is created, it is linked with the current user namespace, and the permissions used to determine if a process can access the resources isolated by the new namespace (network, mount, pid) are the permissions that the process holds in that user namespace.
- if a process has a capability in a particular user namespace, it has that capability in all child user namespaces
- when a new user namespace is created, the creating process is granted all capabilities within that namespace
- the big twist: you don't need special capabilities to create a new user namespace
- high tier, so cool.


S
A
B
C
D
User Namespaces
Network Namespaces
Mount Namespaces
Cgroup Namespaces
PID Namespaces
UTS Namespaces
IPC Namespaces
Time Namespaces

The end, TYVM

- If there’s time, any questions, or experiments you’d like to see?