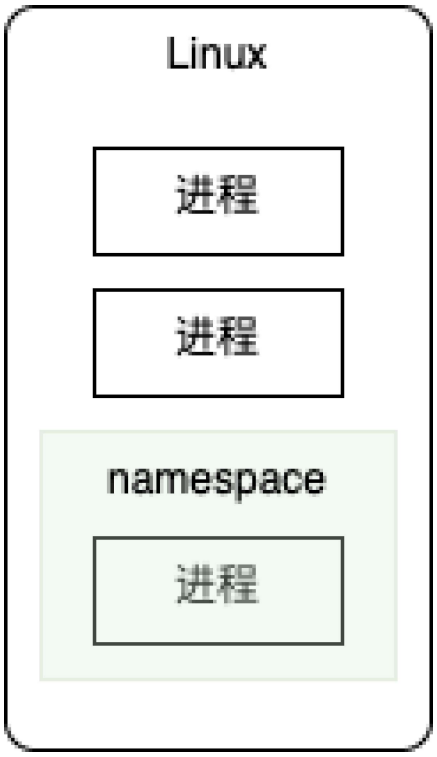
使用Linux Namespace把进程一步一步隔离起来

调度行
写调度器代码还行
+ 关注他

Linux容器技术是一种隔离方式。把进程放到逻辑上的盒子里，让容器里的程序看不到容器外的东西，对于这些进程来说，就好像操作系统里只有他们自己在运行一样。Linux namespaces就是这么一种控制进程该看见什么，不该看见什么的技术。



那么，问题来了，要看不到什么东西，才能让这些进程觉得自己在一个独立的操作系统里面？接下来，在 `Ubuntu 20.04.2 LTS` 操作系统里，一步一步，把进程隔离+起来。

第一步：只看到自己的进程信息而看不到操作系统的其他进程信息。

准备unshare

在继续之前，先准备namespace管理的程序。 `unshare` 创建新的namespace并且在其中运行指定的程序。当然可以直接用root执行 `unshare` ，就不需要这里的麻烦事儿，可以跳过这一步。这里的方法是让任何Linux用户都可以创建namespace，毕竟namespace的设计也没有必要局限在只有特权用户才能使用。

为了保持操作系统干净，从操作系统中将 `unshare` 拷贝到当前目录下。

```
$ cp `which unshare` ./
```

当然，准备的时候，root权限+也是少不了的，一些配置还需要root来做。使用root增加 `unshare` 文件的权能设置：

```
$ setcap 'cap_sys_admin+ep' ./unshare
```

权能系统，Capabilities，是Linux引入的权限控制。系统中不同的操作权限，要配置相对应的权能。比如，要创建pid namespace，就需要cap_sys_admin权限。设置cap_sys_admin到文件的Effective和Permitted capabilities，根据权能规则，使普通用户有权限做cap_sys_admin权能对应的操作。

pid namespace

要只看到自己的pid，需要两步：创建pid namespace，在namespace中挂载 `proc` 文件系统。两个命令行选项，两个操作一次完成。

```
$ unshare --pid --mount-proc --fork bash
$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
lsfadmin       1       0  0 22:36 pts/1    00:00:00 bash
lsfadmin       3       1  0 22:36 pts/1    00:00:00 ps -ef
```

如果只是创建pid namespace，不能保证只看到namespace中的进程。因为类似 `ps` 这类系统工具读取的是 `proc` 文件系统。 `proc` 文件系统没有切换的话，虽然有了pid namespace，但是不能达到我们在这个namespace中只看到属于自己namespace进程的目的。在创建pid namespace的同时，使用 `--mount-proc` 选项，会创建新的mount namespace，并自动 `mount` 新的 `proc` 文件系统。这样， `ps` 就可以看到当前pid namespace里面所有的进程了。因为是新的pid namespace，进程的PID也是从1开始编号。对于pid namespace里面的进程来说，就好像只有自己这些个进程在使用操作系统。

Namepace里的进程，对于本机的操作系统而言，也是一个进程。所以，在操作系统上，也就是namespace之外，也可以看到这个进程，它有属于自己的，操作系统赋予的PID。

```
$ ps -ef | grep unshare
lsfadmin  284459 2714198  0 01:40 pts/1    00:00:00 ./unshare --pid --mount-proc --for
lsfadmin  284801 1858331  0 01:40 pts/2    00:00:00 grep unshare
$ pstree -p 284459
unshare(284459)──bash(284460)
```

两个PID，pid namespace隔离的方式就是这样了。

走到这里，就只看到这个pid namespace里的进程了。感觉上，进程已经隔离开来了。但是不够，进程，比如例子中的 `bash` ，依然可以看到本机操作系统的信息。比如文件系统，比如其中的Ubuntu的版本。

```
$ cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=20.04
DISTRIB_CODENAME=focal
DISTRIB_DESCRIPTION="Ubuntu 20.04.2 LTS"
```

接下来处理文件系统隔离。

mount namespace

为了让进程看不到操作系统的文件系统，只看到属于namespace自己的文件系统，需要创建mount namespace。 `--mount-proc` 的时候，其实就已经创建了新的mount namespace。可是，bash中还是能够看到操作系统的目录和文件。需要达到理想隔离，还是需要一些配合操作的， `mount` 属于这个mount namespace的文件系统。

首先做点准备工作，创建一个根文件系统，一般的容器镜像都可以使用。这里使用docker的ubuntu镜像作为例子。在容器中安装了 `iproute2` ，是为了方便后面network namespace的实验。

```
$ mkdir ubuntu
$ docker run -it ubuntu bash
[container] apt update
[container] apt install iproute2
```

将运行的容器的镜像导出来，解压到Linux当前目录的ubuntu子目录中：

```
$ docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED         STATUS         PORTS     NAMES
3ed040cc2824   ubuntu    ...
$ docker export 3ed040cc2824 --output=ubuntu.tar
$ mkdir ubuntu
$ tar -xf ubuntu.tar -C ubuntu
```

`unshare` 要使用 `chroot` 操作，将namespace中的根文件系统切换，而不影响操作系统。为了让非root用户完成这个操作，额外需要 `cap_sys_chroot` 权能：

```
$ setcap 'cap_sys_admin+ep cap_sys_chroot+ep' ./unshare
```

使用 `unshare` 创建mount namespace，并将系统根目录切换到刚刚创建的根文件系统目录中。这样，namespace中能看到的全部目录，其实只是操作系统中的一个子目录。并且后续的 `mount` 操作，也只会影响mount namespace。

```
$ ./unshare --mount --root /home/lsfadmin/shared/ns/ubuntu --pid --mount-proc --fork b

I have no name!@linux1 :/$ mount
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
I have no name!@linux1 :/$ cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=20.04
DISTRIB_CODENAME=focal
DISTRIB_DESCRIPTION="Ubuntu 20.04.3 LTS"
```

显然，这里新的 `ubuntu` 镜像和操作系统使用的小版本是不一下样的。隔离出来了。原本应该搞一个centos✦之类的镜像，那样看起来区别大一些。

到目前为止，Linux操作系统中的bash和namespace中的bash看到文件系统已经不同。当然，还可以通过在操作系统中执行 `lsns` 命令查看新创建的namespace。

```
$ pstree -p 904489
unshare(904489)──bash(904490)
$ lsns -p 904490
        NS TYPE   NPROCS    PID USER     COMMAND
4026532420 mnt         1 904490 lsfadmin bash
4026532421 pid         1 904490 lsfadmin bash
```

嗯，确实是给 `unshare` 出来的进程创建了新的namespace。

OK。现在像是容器了，只看到容器里的进程，只看到容器里面的文件系统。

user namespace

在user namespace里面，可以自己管理用户的。也就是说，我想是谁就是谁。要做到这一步还是一个组合拳。先创建user namespace：

```
$ ./unshare --user --mount --root /home/lsfadmin/shared/ns/ubuntu --pid --mount-proc -
$ id
uid=65534(nobody) gid=65534(nogroup) groups=6
```

user namespace默认初始化的用户是 `nobody` (65534)。这个时候相当于刚出生，还没起名字(ID)，想叫什么叫什么。让它变成007，把7号ID给自己。在操作系统bash中：

```
$ echo '0 1000 1' > /proc/1904696/uid_map
```

再切回到namespace里面，神奇的事情发生了，007来了。

```
nobody@linux1:/$ id
uid=7(lp) gid=65534(nogroup) groups=65534(nogroup)
```

不错，可以用新身份创建文件了：

```
nobody@linux1:/$ touch from-namespace
nobody@linux1:/$ ls -nl from-namespace
-rw-rw-r-- 1 7 65534 0 Nov 30 14:00 from-namespace
```

第三列，用户7创建文件，如我所料。

但是，事实上，一直以来， `我` 从未改变。Namespace外：

```
$ cd ubuntu
$ ls -nl from-namespace
-rw-rw-r-- 1 1000 1000 0 Nov 30 06:00 from-namespace
```

文件创建的用户依然是 `lsfadmin` ，依然是 `1000` 。操作系统会确保，无论怎样，都是以登陆用户授权访问文件系统。Namespace仅仅是让在那个隔离里面看起来不一样而已。

隔离的越来越多，又向容器里走了一步。其实也可以说已经在容器里面了，因为已然有了隔离。当然可以做的更多，比如还有网络。

network namespace

网络namespace创建后，默认只会创建loopback设备。也就是说，默认只能和自己玩，因为没有连接外界的网络设备。

```
$ ./unshare --net --user --mount --root /home/lsfadmin/shared/ns/ubuntu --pid --mount-
nobody@linux1:/$ ip addr
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

想要网络玩起来，那就是另一个故事 `ip netns` ✦ 。

UTS namespace

从shell的提示符中，还可以看到本机操作系统的影子 `linux1` ，主机操作系统设置的机器名字。shell在启动的时候，会用主机机器名生成这个提示符。通过命令行，也可以看到确实是本机的机器名。

```
$ ./unshare --uts --net --user --mount --root /home/lsfadmin/shared/ns/ubuntu --pid --
nobody@linux1:/$ hostname
linux1
nobody@linux1:/$ hostname hello
hostname: you must be root to change the host name
```

改不了！怎么回事儿？在uts namespace里面，自己还不能更改机器名？

在namespace里面，也有权限的管理。系统配置的改动，也是遵循类似操作系统中的规则的。Linux操作系统中，要改动机器名，要首先获取特权用户权限，root。这也是合理的，因为这样，可以在namespace中模拟出操作系统的行为，让容器作为一个独立操作系统看待，更体现虚拟化这个概念。在UTS namespace里面，要改机器名，也做类似检查，需要root，这个root是namespace中的root。当然，它的影响也仅仅局限在namespace里面。使用user namespace中相同的方法，只是映射用户ID到0而不是7。

```
nobody@linux:/$ id
uid=0(root) gid=65534(nogroup) groups=65534(nogroup)
nobody@linux1:/$ hostname hello
nobody@linux1:/$ hostname
hello
```

更改成功。

如果还要纠结为什么提示符没有改过来，虽然与主题无关，还是解释一下，因为shell在初始化的时候设置一次提示符。如果想要机器名在提示符也生效，只需要再启动一个shell就成了。如果前面对于user namespace有类似的疑问，也是同样的答案：bash需要初始化，读取系统配置，来改变提示符中的用户名。

```
nobody@linux1:/$ bash
root@hello:/#
```

IPC namespace

Linux中的很多操作都是使用文件进行的，所以文件模式的进程间通信+只要mount namespace将文件从本机操作系统 `mount` 进来应该就可以了玩了(没玩过，逻辑上对)。但是Linux进程间通信的实现不是统一的，一些实现不可以：System V的IPC对象和POSIX的消息队列。使用POSIX消息队列作为例子演示隔离。本机操作系统中创建一个消息队列。

```
lsfadmin@linux1 $ ipcmk --queue
Message queue id: 1
lsfadmin@linux1 $ ipcs --queues

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x7f247413 1          lsfadmin   644        0            0
```

创建新的消息队列，隔离效果立刻显现：没了。

```
$ ./unshare --ipc --uts --net --user  --mount --root /home/lsfadmin/shared/ns/ubuntu -
nobody@linux1:/$ ipcs --queues

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

nobody@linux1:/$
```

当然，自己namespace创建的对象是namespace内部可见的。

容器

到这里，我们已经得到了一个独立的环境，bash已经在namespace创建的盒子里了。看起来像是拥有了自己的操作系统一样：自己的机器名，自己的用户系统，自己的文件系统，自己的网络，自己的进程们PID，自己进程彼此之间的IPC。这基本上已经算是一个虚拟的操作系统了。

```
$ ./unshare --ipc --uts --net --user --mount --root /home/lsfadmin/shared/ns/ubuntu --
```

如果你想要一个自己的容器实现，写一个脚本，把文章中的步骤自动化，就可以完成一个简单容器技术。 当你想要更底层的实现，打开unshare.c，参考它重写你的容器实现逻辑，也不是一件复杂的事情。

当然，容器使用了namespace，但容器不仅仅是namespace。

未完待续...

参考

- man chroot: chroot是一个很有历史的系统调用，它原本是独立出现的，也被独立使用，用来隔离根文件系统。随着mount namespac的引用，彼此有了配合。
- man namespaces 了解namespace的概念
- man unshare 了解unshare的用法
- man *anything* 了解你不知道的命令和概念

编辑于 2021-12-05 07:48

内容所属专栏

进程 Linux 开发 Linux