Linux Namespace : 简介

在初步的了解 docker 后，笔者期望通过理解 docker 背后的技术原理来深入的学习和使用 docker，接下来的几篇文章简单的介绍下 linux namespace 的概念以及基本用法。

namespace 的概念

namespace 是 Linux 内核用来隔离内核资源的方式。通过 namespace 可以让一些进程只能看到与自己相关的一部分资源，而另外一些进程也只能看到与它们自己相关的资源，这两拨进程根本就感觉不到对方的存在。具体的实现方式是把一个或多个进程的相关资源指定在同一个 namespace 中。

Linux namespaces 是对全局系统资源的一种封装隔离，使得处于不同 namespace 的进程拥有独立的全局系统资源，改变一个 namespace 中的系统资源只会影响当前 namespace 里的进程，对其他 namespace 中的进程没有影响。

namespace 的用途

可能绝大多数的使用者和我一样，是在使用 docker 后才开始了解 linux 的 namespace 技术的。实际上，Linux 内核实现 namespace 的一个主要目的就是实现轻量级虚拟化(容器)服务。在同一个 namespace 下的进程可以感知彼此的变化，而对外界的进程一无所知。这样就可以让容器中的进程产生错觉，认为自己置身于一个独立的系统中，从而达到隔离的目的。也就是说 linux 内核提供的 namespace 技术为 docker 等容器技术的出现和发展提供了基础条件。
我们可以从 docker 实现者的角度考虑该如何实现一个资源隔离的容器。比如是不是可以通过 chroot 命令切换根目录的挂载点，从而隔离文件系统。为了在分布式的环境下进行通信和定位，容器必须要有独立的 IP、端口和路由等，这就需要对网络进行隔离。同时容器还需要一个独立的主机名以便在网络中标识自己。接下来还需要进程间的通信、用户权限等的隔离。最后，运行在容器中的应用需要有进程号(PID)，自然也需要与宿主机中的 PID 进行隔离。也就是说这六种隔离能力是实现一个容器的基础，让我们看看 linux 内核的 namespace 特性为我们提供了什么样的隔离能力：

上表中的前六种 namespace 正是实现容器必须的隔离技术，至于新近提供的 Cgroup namespace 目前还没有被 docker 采用。相信在不久的将来各种容器也会添加对 Cgroup namespace 的支持。

namespace 的发展历史

Linux 在很早的版本中就实现了部分的 namespace，比如内核 2.4 就实现了 mount namespace。大多数的 namespace 支持是在内核 2.6 中完成的，比如 IPC、Network、PID、和 UTS。还有个别的 namespace 比较特殊，比如 User，从内核 2.6 就开始实现了，但在内核 3.8 中才宣布完成。同时，随着 Linux 自身的发展以及容器技术持续发展带来的需求，也会有新的 namespace 被支持，比如在内核 4.6 中就添加了 Cgroup namespace。

Linux 提供了多个 API 用来操作 namespace，它们是 clone()、setns() 和 unshare() 函数，为了确定隔离的到底是哪项 namespace，在使用这些 API 时，通常需要指定一些调用参数：CLONE_NEWIPC、CLONE_NEWNET、CLONE_NEWNS、CLONE_NEWPID、CLONE_NEWUSER、CLONE_NEWUTS 和 CLONE_NEWCGROUP。如果要同时隔离多个 namespace，可以使用 | (按位或)组合这些参数。同时我们还可以通过 /proc 下面的一些文件来操作 namespace。下面就让让我们看看这些接口的简要用法。

查看进程所属的 namespace

从版本号为 3.8 的内核开始，/proc/[pid]/ns 目录下会包含进程所属的 namespace 信息，使用下面的命令可以查看当前进程所属的 namespace 信息：

```
$ ll /proc/$$/ns
```

首先，这些 namespace 文件都是链接文件。链接文件的内容的格式为 xxx:[inode number]。其中的 xxx 为 namespace 的类型，inode number 则用来标识一个 namespace，我们也可以把它理解为 namespace 的 ID。如果两个进程的某个 namespace 文件指向同一个链接文件，说明其相关资源在同一个 namespace 中。
其次，在 /proc/[pid]/ns 里放置这些链接文件的另外一个作用是，一旦这些链接文件被打开，只要打开的文件描述符(fd)存在，那么就算该 namespace 下的所有进程都已结束，这个 namespace 也会一直存在，后续的进程还可以再加入进来。
除了打开文件的方式，我们还可以通过文件挂载的方式阻止 namespace 被删除。比如我们可以把当前进程中的 uts 挂载到 ~/uts 文件：

```
$ touch ~/uts
$ sudo mount --bind /proc/$$/ns/uts ~/uts
```

使用 stat 命令检查下结果：

很神奇吧，~/uts 的 inode 和链接文件中的 inode number 是一样的，它们是同一个文件。

clone() 函数

我们可以通过 clone() 在创建新进程的同时创建 namespace。clone() 在 C 语言库中的声明如下：

```
/* Prototype for the glibc wrapper function */
#define _GNU_SOURCE
#include <sched.h>
int clone(int (*fn)(void *), void *child_stack, int flags, void *arg);
```

实际上，clone() 是在 C 语言库中定义的一个封装(wrapper)函数，它负责建立新进程的堆栈并且调用对编程者隐藏的 clone() 系统调用。Clone() 其实是 linux 系统调用 fork() 的一种更通用的实现方式，它可以通过 flags 来控制使用多少功能。一共有 20 多种 CLONE_ 开头的 falg(标志位) 参数用来控制 clone 进程的方方面面(比如是否与父进程共享虚拟内存等)，下面我们只介绍与 namespace 相关的 4 个参数：

- fn：指定一个由新进程执行的函数。当这个函数返回时，子进程终止。该函数返回一个整数，表示子进程的退出代码。
- child_stack：传入子进程使用的栈空间，也就是把用户态堆栈指针赋给子进程的 esp 寄存器。调用进程(指调用 clone() 的进程)应该总是为子进程分配新的堆栈。
- flags：表示使用哪些 CLONE_ 开头的标志位，与 namespace 相关的有CLONE_NEWIPC、CLONE_NEWNET、CLONE_NEWNS、CLONE_NEWPID、CLONE_NEWUSER、CLONE_NEWUTS 和 CLONE_NEWCGROUP。
- arg：指向传递给 fn() 函数的参数。

在后续的文章中，我们主要通过 clone() 函数来创建并演示各种类型的 namespace。

setns() 函数

通过 setns() 函数可以将当前进程加入到已有的 namespace 中。setns() 在 C 语言库中的声明如下：

```
#define _GNU_SOURCE
#include <sched.h>
int setns(int fd, int nstype);
```

和 clone() 函数一样，C 语言库中的 setns() 函数也是对 setns() 系统调用的封装：

- fd：表示要加入 namespace 的文件描述符。它是一个指向 /proc/[pid]/ns 目录中文件的文件描述符，可以通过直接打开该目录下的链接文件或者打开一个挂载了该目录下链接文件的文件得到。
- nstype：参数 nstype 让调用者可以检查 fd 指向的 namespace 类型是否符合实际要求。若把该参数设置为 0 表示不检查。

前面我们提到：可以通过挂载的方式把 namespace 保留下来。保留 namespace 的目的是为以后把进程加入这个 namespace 做准备。在 docker 中，使用 docker exec 命令在已经运行着的容器中执行新的命令就需要用到 setns() 函数。为了把新加入的 namespace 利用起来，还需要引入 execve() 系列的函数(笔者在《Linux 创建子进程执行任务》一文中介绍过 execve() 系列的函数，有兴趣的同学可以前往了解)，该函数可以执行用户的命令，比较常见的用法是调用 /bin/bash 并接受参数运行起一个 shell。

unshare() 函数 和 unshare 命令

通过 unshare 函数可以在原进程上进行 namespace 隔离。也就是创建并加入新的 namespace 。unshare() 在 C 语言库中的声明如下：

```
#define _GNU_SOURCE
#include <sched.h>
int unshare(int flags);
```

最新文章
svn处理文件冲突方法
svn 提交代码
react 入坑学习(八)Hooks React 父组...
Ant Design Form表单中getFieldDeco...
js中如何将object转化为json数据. jso...
Http请求
HttpServletResponse(set方法)
ServletContext对象
ServletRequest和ServletResponse
JavaWeb核心之Servlet

热门文章
XML--模拟servlet执行
元类
socket(套接字)
文件exer1的访问权限为rw-r--r--,现要...
什么是链接？
查看后台进程作业ID的指令
用户态切换到内核态的 3 种方式
linux osi 层
linux 读写权限
linux

和前面两个函数一样，C 语言库中的 unshare() 函数也是对 unshare() 系统调用的封装。调用 unshare() 的主要作用就是：不启动新的进程就可以起到资源隔离的效果，相当于跳出原先的 namespace 进行操作。

系统还默认提供了一个叫 unshare 的命令，其实就是在调用 unshare() 系统调用。下面的 demo 使用 unshare 命令把当前进程的 user namespace 设置成了 root：

总结

namespace 是 linux 内核提供的特性，为虚拟化而生。随着 docker 的诞生引爆了容器技术，也把长期在后台默默奉献的 namespace 技术推到了大家的面前。笔者试图通过对 namespace 技术的学习和理解来加深对容器技术的认识，所以接下来会通过文章记录学习 namespace 的点点滴滴，希望能和同学们一起进步。

参考：

Namespace 概述
overview of Linux namespaces
Clone 函数
Setns 函数
Unshare 函数

相关阅读:

显示内容和隐藏v-show(以及图标的动态展示)
主表查询子表
怎么在pda安装apk
java学习第40天2020/8/14
Java学习第39天2020/8/13
java学习第38天2020/8/12
java学习第37天2020/8/11
rz
git tag
audio vedio 播放