02-Namespace详解

Namespace简介

Linux Namespace是Linux提供的一种内核级别环境隔离的方法。很早以前的Unix有一个叫chroot的系统调用（通过修改根目录把用户jail到一个特定目录下），chroot提供了一种简单的隔离模式：chroot内部的文件系统无法访问外部的内容。Linux Namespace在此基础上，提供了对UTS、IPC、mount、PID、network、User等的隔离机制。

举个例子，我们都知道，Linux下的超级父亲进程的PID是1，所以，同chroot一样，如果我们可以把用户的进程空间jail到某个进程分支下，并像chroot那样让其下面的进程 看到的那个超级父进程的PID为1，于是就可以达到资源隔离的效果了（不同的PID namespace中的进程无法看到彼此）

分类	系统调用参数	相关内核版本
Mount namespaces	CLONE_NEWNS	Linux 2.4.19
UTS namespaces	CLONE_NEWUTS	Linux 2.6.19
IPC namespaces	CLONE_NEWIPC	Linux 2.6.19
PID namespaces	CLONE_NEWPID	Linux 2.6.24
Network namespaces	CLONE_NEWNET	始于Linux 2.6.24 完成于 Linux 2.6.29
User namespaces	CLONE_NEWUSER	始于 Linux 2.6.23 完成于 Linux 3.8)

主要是三个系统调用

- **`clone****()` – 实现线程的系统调用，用来创建一个新的进程，并可以通过设计上述参数达到隔离。
- **`unshare****()` – 使某进程脱离某个namespace
- **`setns****()` – 把某进程加入到某个namespace

unshare() 和 setns() 都比较简单，大家可以自己man，这里不说了。

(注：Docker使用命令docker exec 进入到某个容器内使用的setns这样的系统调用)

clone()系统调用

我们来看一下一个最简单的clone()系统调用的示例：

```
#define _GNU_SOURCE
#include <sys/mount.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <sched.h>
#include <signal.h>
#include <unistd.h>
#define STACK_SIZE (1024 * 1024)
static char container_stack[STACK_SIZE];
char* const container_args[] = {
  "/bin/bash",
  NULL
};

int container_main(void* arg)
{
  printf("Container - inside the container!\n");
  execv(container_args[0], container_args);
  printf("Something's wrong!\n");
  return 1;
}

int main()
{
  printf("Parent - start a container!\n");
  int container_pid = clone(container_main, container_stack+STACK_SIZE, SIGCHLD , NULL);
  waitpid(container_pid, NULL, 0);
  printf("Parent - container stopped!\n");
  return 0;
}
```

这段代码的功能非常简单：在 main 函数里，我们通过 clone() 系统调用创建了一个新的子进程 container_main,执行了一个/bin/bash，

但是对于上面的程序，父子进程的进程空间是没有什么差别的，父进程能访问到的子进程也能。

下面， 让我们来看几个例子看看，Linux的Namespace是什么样的。

```
#define _GNU_SOURCE
#include <sys/mount.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <sched.h>
#include <signal.h>
#include <unistd.h>
#define STACK_SIZE (1024 * 1024)
static char container_stack[STACK_SIZE];
char* const container_args[] = {
  "/bin/bash",
  NULL
};
int container_main(void* arg)
{
  printf("Container - inside the container!\n");
  sethostname("test",10); /* hostname */
  execv(container_args[0], container_args);
  printf("Something's wrong!\n");
  return 1;
}
int main()
{
  printf("Parent - start a container!\n");
  int container_pid = clone(container_main, container_stack+STACK_SIZE, CLONE_NEWUTS | CLONE_NEWPID | SIGCHLD ,
NULL);
  waitpid(container_pid, NULL, 0);
  printf("Parent - container stopped!\n");
  return 0;
}
```

上面的clone系统调用中，我们新增了2个参数：CLONE_NEWUTS、CLONE_NEWPID；也就是新增了UTS和PID两种Namespace，

编译执行上面的程序，发现子进程的hostname变成了 container，同时查看当前的进程号变成了1：

```
$ vi test.c
$ gcc -o test test.c
$ ./test
Parent - start a container!
Container - inside the container!
$ hostname
test
$ echo $$
1
```

我们知道，在传统的UNIX系统中，PID为1的进程是init，地位非常特殊。他作为所有进程的父进程，有很多特权（比如：屏蔽信号等），另外，其还会为检查所有进程的状态，如果某个子进程脱离了父进程（父进程没有wait它），那么init就会负责回收资源并结束这个子进程。所以，要做到进程空间的隔离，首先要创建出PID为1的进程，最好就像chroot那样，把子进程的PID在容器内变成1。

但是，我们会发现，在子进程的shell里输入ps,top等命令，我们还是可以看得到所有进程。说明并没有完全隔离。这是因为，像ps, top这些命令会去读/proc文件系统，所以，因为/proc文件系统在父进程和子进程都是一样的，所以这些命令显示的东西都是一样的。

所以，我们还需要对文件系统进行隔离。

Mount Namespace隔离

下面的例程中，我们在启用了mount namespace并在子进程中重新mount了/proc文件系统：

```
#define _GNU_SOURCE
#include <sys/mount.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <sched.h>
#include <signal.h>
#include <unistd.h>
#define STACK_SIZE (1024 * 1024)
static char container_stack[STACK_SIZE];
char* const container_args[] = {
  "/bin/bash",
  NULL
};
int container_main(void* arg)
{
    printf("Container [%5d] - inside the container!\n", getpid());
    sethostname("test",10);
    /* mount proc /proc */
    system("mount -t proc proc /proc");
    execv(container_args[0], container_args);
    printf("Something's wrong!\n");
    return 1;
}
int main()
{
    printf("Parent [%5d] - start a container!\n", getpid());
    /* Mount Namespace - CLONE_NEWNS */
    int container_pid = clone(container_main, container_stack+STACK_SIZE,
            CLONE_NEWUTS | CLONE_NEWPID | CLONE_NEWNS | SIGCHLD, NULL);
    waitpid(container_pid, NULL, 0);
    printf("Parent - container stopped!\n");
    return 0;
}
```

运行结果如下：

```
./mount
Parent [16436] - start a container!
Container [    1] - inside the container!
$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 11:25 pts/0    00:00:00 /bin/bash
root          10       1  0 11:25 pts/0    00:00:00 ps -ef
```

上面，我们可以看到只有两个进程 ，而且pid=1的进程是我们的/bin/bash。我们还可以看到/proc目录下也干净了很多：

```
ls /proc
1     buddyinfo  consoles  diskstats    fb           iomem     kcore      kpagecgroup  locks    modules
pagetypeinfo  schedstat  softirqs  sysrq-trigger  tty              vmallocinfo
15    bus        cpuinfo   dma          filesystems  ioports   keys       kpagecount   mdstat   mounts
partitions    scsi       stat      sysvipc        uptime           vmstat
23    cgroups    crypto    driver       fs           irq       key-users  kpageflags   meminfo  mtrr
pressure      self       swaps     thread-self    version          zoneinfo
acpi  cmdline    devices   execdomains  interrupts   kallsyms  kmsg       loadavg      misc     net
sched_debug   slabinfo   sys       timer_list     version_signature
```

在通过CLONE_NEWNS创建mount namespace后，父进程会把自己的文件结构复制给子进程中。而子进程中新的namespace中的所有mount操作都只影响自身的文件系统，而不对外界产生任何影响，这样可以做到比较严格地隔离。

事实上，作为一个普通用户，我们希望的情况是：每当创建一个新容器时，容器进程看到的文件系统就是一个独立的隔离环境，而不是继承自宿主机的文件系统，这里可以通过容器进程启动之前重新挂载它的整个根目录“/”。而由于 Mount Namespace 的存在，这个挂载对宿主机不可见，所以容器进程就可以在里面随便折腾了。

在 Linux 操作系统里，有一个名为 chroot 的命令可以帮助你在 shell 中方便地完成这个工作。它的作用就是帮你“change root file system”，即改变进程的根目录到你指定的位置。它的用法也非常简单。

假设，我们现在有一个 $HOME/test 目录，想要把它作为一个 /bin/bash 进程的根目录。

首先，创建一个 test 目录和几个 lib 文件夹：

```
$ mkdir -p $HOME/test/{bin,lib64,lib}
```

然后，把 bash 命令拷贝到 test 目录对应的 bin 路径下：

```
$ cp -v /bin/{bash,ls} $HOME/test/bin
```

接下来，把 bash 命令需要的所有 so 文件，也拷贝到 test 目录对应的 lib 路径下：

```
$ T=$HOME/test
$ list="$(ldd /bin/ls | egrep -o '/lib.*\.[0-9]')"
$ for i in $list; do cp -v "$i" "${T}${i}"; done
```

最后，执行 chroot 命令，告诉操作系统，我们将使用 $HOME/test 目录作为 /bin/bash 进程的根目录：

```
$ chroot $HOME/test /bin/bash
```

这时，如果执行 "ls /"，就会看到，它返回的都是 $HOME/test 目录下面的内容，而不是宿主机的内容。

更重要的是，对于被 chroot 的进程来说，它并不会感受到自己的根目录已经被“修改”成 $HOME/test 了。

实际上，Mount Namespace 正是基于对 chroot 的不断改良才被发明出来的，它也是 Linux 操作系统里的第一个 Namespace。

当然，为了能够让容器的这个根目录看起来更“真实”，一般会在这个容器的根目录下挂载一个完整操作系统的文件系统，比如 Ubuntu18.04 的 ISO。这样，在容器启动之后，我们在容器里通过执行 "ls /" 查看根目录下的内容，就是 Ubuntu 18.04 的所有目录和文件。

而这个挂载在容器根目录上、用来为容器进程提供隔离后执行环境的文件系统，就是我们下一节准备说的“容器镜像”