Assignment 3 - Simple Resource Container
COMP 310 - Operating Systems
Prof. Muthucumaru Maheswaran
Fall 2018

LE, Nhat Hung
McGill ID: 260793376
Date: November 26, 2018
Due date: November 27, 2018

Create and enter docker container:
```
docker run --privileged -d --rm --name=hle30_260793376_container
smean_image_comp310
docker exec -it hle30_260793376_container /bin/bash
```

Useful commands:
`kill -9 - Kill a process`
`htop -u - To list processes run by you`

Print out the process tree: `pstree -g -s`
Linux's 7 namespaces: `PID, IPC, UTS, NET, USER, CGROUP, and MNT`
Put the shell in its own PID namespace (the -p option): `sudo unshare -fp /bin/sh`

To confine the process listing to the ones in the newly created namespace, we need to mount the proc file system again: `sudo unshare -fp --mount-proc=/proc /bin/bash`

Unshare w PID + USER namespaces: `unshare -fpU --mount-proc=/proc /bin/bash`
W PID + UTS namespaces: `unshare -fpu --mount-proc=/proc /bin/bash`
PID of current shell: `echo $$`
User of current shell: `id`

Upload a mapping between the root (0) in the parent namespace and child namespace:
`newuidmap <PID of unshare process> 0 0 1`

Same for group: `newgidmap <PID of unshare process> 0 0 1`

Print hostname: `hostname`
Modify hostname: `hostname <new name>`

Download file systems:
`debootstrap jessie jrootfs` (Gets a minimalistic file-system from Debian-jessie)
`debootstrap stretch srootfs` (Gets a minimalistic file-system from Debian-stretch)

Create file system isolated from host:
```
cd jrootfs
sudo unshare -fpu --mount-proc=/proc /bin/bash
chroot .
```

In new container, mount proc file system: `mount -t proc proc /proc`

Recursively bind mount: `mount --rbind /sys/fs/cgroup <fs path>/sys/fs/cgroup`

Creating memory limiter:
`cd /sys/fs/cgroup/memory`

```
mkdir ExMemLimiter
cd ExMemLimiter
```

Add container to memory cgroup (control group):

```
echo 0 > tasks
```

Creating CPU usage limiter:

```
cd /sys/fs/cgroup/cpu
mkdir CPULimiter
cd CPULimiter
echo <integer> > cpu.shares
```

Restrict both containers to CPU-core-0 with 2 memory nodes:

```
cd /sys/fs/cgroup/cpuset
mkdir CPURestrictor
cd CPURestrictor
echo 0 > cpuset.cpus
echo 0-1 > cpuset.mems
```

Stress CPU:

```
apt-get install stress
stress --cpu 1 v
```

1. Experiment each step mentioned in the handout and get a clear understanding as to what is happening. (ex: what exactly is unshare doing?)

> i. Shells in separate namespaces can't see each other's processes.
> ii. Host can see processes in other namespaces using htop, but not using pstree.
> iii. Can kill from host, but not from other namespaces created through unshare in this same host.
> iv. Shells in created namespaces can't see host's processes.
>
> unshare basically creates an unshared container, with its own namespace or namespaces. Is pretty much a virtual machine/environment. Useful for development: Dockerfile prime example.

2. After mounting the proc subsystem into the unshared-container, see the difference between the proc folder inside and outside and see the difference.

> Inside container, only 2 PIDs: 1 and a PID that increments when a command is run
> Outside container (inside host), all PIDs, including the real PID of the container is listed.

3. Why is it necessary to mount "proc" to be able to get commands like top and ps working inside the container?

> The command is getting the process listing information from the /proc file system interface. So, we are seeing all the processes instead of confining the output to the newly created namespace that is holding the shell process. To confine the process listing to the ones in the newly created namespace, we need to mount the proc file system again.

4. How did you validate that the container was in a new PID namespace?

"`echo $$`" inside container shows PID = 1.

Run two containers in two separate namespaces. Run some commands in the background in one namespace.

a. `ping 8.8.8.8 /dev/null &` (3 times to run three instances of ping)

Now get the PIDs of these 3 ping processes. Try killing them inside the other container using:

```
kill -9 PID
```

Do you see the processes from the host?

Process attempting kill thinks target does not exist. "htop" doesn't see processes running in other container.
The host, however, does see the processes.

5. Create a new user in your host

```
sudo adduser newt
```

Change user to this new_user and run an unshared shell with isolated USER namespace. Now set map the uid of this new-user to some random value inside the namespace.

No values other than "`<new user id> <new user id> 1`" are allowed.
Only same user (owner of container/namespace) is allowed to make change.

6. Run two separate containers in different UTS namespace. Set their hostnames to different things and see if changing one affects the other.

Containers can't set each other's hostname.

7. Without chroot, can you traverse anywhere within the host filesystem?

Yes.

8. Now run two containers and chroot into the same root-filesystem. Create/Delete files in container and see if they are visible in the other.

Yes. Exactly because they are in the same root file system. It's possible to make a copy of the file system. `cd` there then `chroot .` . Then the two containers will have isolated file systems.

9. Now create two copies of root-filesystems. Run two containers with chroot'ed to different roots. See if you can traverse to the filesystem of the other container.

Impossible. Unless exit from chroot command.

10. Create a container, try running a memory hogging program and view its memory usage using htop.

Mem hogger runs until finish.

11. Now set a memory-control on this container and run a similar program and see what happens.

    Process killed after time's passed.

12. Create two separate containers and make them share one CPU on the ratio of 7:3.

```
echo 7 > cpu.shares /* in CPULimiter */
echo 3 > cpu.shares /* in CPULimiter2 */
```

13. Restrict the read bytes per second for a container to 10Mbps using a blkio controller

```
echo 10000000 > blkio.throttle.read_bps_device
```

14. Restrict the write bytes per second for a container to 5Mbps using a blkio controller

```
echo 5000000 > blkio.throttle.write_bps_device
```

15. Set a pid-controller to your container so that it can only create 50 processes at max

```
mkdir /sys/fs/cgroup/pids/PROCLIMITER
cd /sys/fs/cgroup/pids/PROCLIMITER
echo 50 > pids.max
echo 0 > tasks
```