1.1 Namespace 和 Cgroups

1.1.1 Namespace

步骤 1　运行一个 centos 容器，并设置容器的 hostname=hwhost。

```
docker run -h hwhost -it centos
```

```
[root@localhost ~]# docker run -h hwhost -it centos
[root@hwhost /]#
```

步骤 2　在容器中添加一个用户 hwusera。

```
useradd hwusera
```

```
[root@hwhost /]# useradd hwusera
[root@hwhost /]#
[root@hwhost /]# su hwusera
[hwusera@hwhost /]$
```

步骤 3　打开另一个宿主机终端，设置宿主机 hostname=HUAWEI。

```
hostname HUAWEI
```

```
[root@localhost /]# hostname HUAWEI
[root@localhost /]# hostname
HUAWEI
[root@localhost /]# su hwusera
su: user hwusera does not exist
```

通过以上可知，容器进程的 user 空间和 hostname 空间分别由 user namespace 和 UTS namespace 隔离，独立于宿主机。

步骤 4　在容器中查看 PID，/bin/bash 的 PID=1。

```
ps axf
```

```
[hwusera@hwhost /]$ ps axf
   PID TTY      STAT   TIME COMMAND
     1 pts/0    Ss     0:00 /bin/bash
    63 pts/0    S      0:00 su hwusera
    64 pts/0    S      0:00  \_ bash
    79 pts/0    R+     0:00      \_ ps axf
```

步骤 5　在宿主机上查看该容器的 PID，/bin/bash 的 PID=141944。

```
ps axf
```

相关回显

```
175586 ?        Ssl    6:22  \_ docker-containerd --config /var/run/docker/containerd/containerd.toml
141930 ?        Sl     0:00    \_ docker-containerd-shim -namespace moby -workdir /var/lib/docker/containerd/daemon/io.containerd.runtime.v1.linux/moby/09dec586d
141944 pts/0    Ss     0:00        \_ /bin/bash
143754 pts/0    S      0:00            \_ su hwusera
143755 pts/0    S+     0:00                \_ bash
```

通过以上可知，容器进程的 PID 空间由 PID namespace 隔离，独立于宿主机。

步骤 6 为方便后续实验，删除本小节中的容器。

```
docker kill
docker rm
```

1.1.2 CPU 资源限制

步骤 1 启动一个名为 huawei1 的压力测试容器，CPU 权重设置为 512。

```
[root@localhost ~]# docker run --name huawei1 -it --cpu-shares 512
progrium/stress --cpu 2
```

```
[root@localhost ~]# docker run --name huawei1 -it --cpu-shares 512 progrium/stress --cpu 2
stress: info: [1] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd
stress: dbug: [1] using backoff sleep of 6000us
stress: dbug: [1] --> hogcpu worker 2 [6] forked
stress: dbug: [1] using backoff sleep of 3000us
stress: dbug: [1] --> hogcpu worker 1 [7] forked
```

步骤 2 打开第二个宿主机终端，TOP 查看 CPU 使用率，已将近 100%。

```
top
```

```
top - 21:11:48 up 17:50,  4 users,  load average: 3.02, 3.90, 3.94
Tasks: 126 total,   3 running, 123 sleeping,   0 stopped,   0 zombie
%Cpu(s):100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  7970316 total,  7140620 free,   250616 used,   579080 buff/cache
KiB Swap:        0 total,        0 free,        0 used.  7326644 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
50250 root      20   0    7304     96      0 R  99.7  0.0  10:12.43 stress
50251 root      20   0    7304     96      0 R  99.7  0.0  10:09.74 stress
 1323 root      20   0  475828  30456  12420 S   0.3  0.4   2:40.24 docker-containe
    1 root      20   0  128044   6680   4172 S   0.0  0.1   0:31.62 systemd
```

虽然设置了权重值，但 CPU 已经到 100%。

步骤 3 打开第三个宿主机终端，启动第二个压力测试容器 huawei2，权重指设置为 2048。

```
[root@localhost ~]# docker run --name huawei2 -it --cpu-shares 2048
progrium/stress --cpu 2
```

```
[root@localhost ~]# docker run --name huawei2 -it --cpu-shares 2048 progrium/stress --cpu 2
stress: info: [1] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd
stress: dbug: [1] using backoff sleep of 6000us
stress: dbug: [1] --> hogcpu worker 2 [6] forked
stress: dbug: [1] using backoff sleep of 3000us
stress: dbug: [1] --> hogcpu worker 1 [7] forked
```

步骤 4 再次查看 CPU 使用率。huawei2 的 CPU 占用将近是 huawei1 的 4 倍。

```
top
```

```
top - 21:14:40 up 17:53,  4 users,  load average: 4.19, 3.85, 3.90
Tasks: 131 total,   5 running, 126 sleeping,   0 stopped,   0 zombie
%Cpu(s): 99.7 us,  0.3 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  7970316 total,  7120644 free,   265972 used,   583700 buff/cache
KiB Swap:        0 total,        0 free,        0 used.  7306840 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
52494 root      20   0    7304     96      0 R  80.4  0.0   1:14.39 stress
52493 root      20   0    7304     96      0 R  78.1  0.0   1:14.89 stress
50251 root      20   0    7304     96      0 R  19.3  0.0  11:44.91 stress
50250 root      20   0    7304     96      0 R  18.9  0.0  11:48.21 stress
 1323 root      20   0  475828  30456  12420 S   0.7  0.4   2:40.59 docker-containe
    1 root      20   0  128044   6680   4172 S   0.3  0.1   0:31.72 systemd
```

步骤 5　查看 CONTAINER ID。

```
docker ps
```

```
[root@localhost docker]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
d2fc304f3563        progrium/stress     "/usr/bin/stress --v…"   27 seconds ago      Up 25 seconds                           huawei2
6597fd9b1507        progrium/stress     "/usr/bin/stress --v…"   5 minutes ago       Up 5 minutes                            huawei1
```

步骤 6　根据 CONTAINER ID，查找 cgroup 下相应的 CPU 配置文件。此处以 huawei2 为例。

```
[root@localhost d2fc304f3563573562a7ccd3dd4d4bcb23f7c9dfede0e352427b9ff5808b12e0]# pwd
/sys/fs/cgroup/cpu/docker/d2fc304f3563573562a7ccd3dd4d4bcb23f7c9dfede0e352427b9ff5808b12e0
[root@localhost d2fc304f3563573562a7ccd3dd4d4bcb23f7c9dfede0e352427b9ff5808b12e0]# ll
total 0
-rw-r--r--. 1 root root 0 Aug 13 21:22 cgroup.clone_children
--w--w--w-. 1 root root 0 Aug 13 21:22 cgroup.event_control
-rw-r--r--. 1 root root 0 Aug 13 21:22 cgroup.procs
-r--r--r--. 1 root root 0 Aug 13 21:22 cpuacct.stat
-rw-r--r--. 1 root root 0 Aug 13 21:22 cpuacct.usage
-r--r--r--. 1 root root 0 Aug 13 21:22 cpuacct.usage_percpu
-rw-r--r--. 1 root root 0 Aug 13 21:22 cpu.cfs_period_us
-rw-r--r--. 1 root root 0 Aug 13 21:22 cpu.cfs_quota_us
-rw-r--r--. 1 root root 0 Aug 13 21:22 cpu.rt_period_us
-rw-r--r--. 1 root root 0 Aug 13 21:22 cpu.rt_runtime_us
-rw-r--r--. 1 root root 0 Aug 13 21:22 cpu.shares
-r--r--r--. 1 root root 0 Aug 13 21:22 cpu.stat
-rw-r--r--. 1 root root 0 Aug 13 21:22 notify_on_release
-rw-r--r--. 1 root root 0 Aug 13 21:22 tasks
```

步骤 7　查看 cpu.shares 值和 tasks 值。cpu.shares=2048，tasks 值即 huawei2 在宿主机上的 PID。

```
cat cpu.shares
cat tasks
```

```
[root@localhost d2fc304f3563573562a7ccd3dd4d4bcb23f7c9dfede0e352427b9ff5808b12e0]# cat cpu.shares
2048
[root@localhost d2fc304f3563573562a7ccd3dd4d4bcb23f7c9dfede0e352427b9ff5808b12e0]# cat tasks
217401
217429
```

步骤 8　为方便后续实验，删除本小节中的容器。

```
docker kill
docker rm
```

1.1.3 内存资源限制

步骤 1 在后台启动一个 centos 容器，并限制其最多使用 400M 内存和 100M swap。

```
docker run -m 400M --memory-swap=500M -dit centos /bin/bash
```

```
[root@localhost ~]# docker run -m 400M --memory-swap=500M -dit centos /bin/bash
515da3dcef3ced12e714f66b3224f1f73588ab0b8eaa630166e8e6b955608911
```

步骤 2 找到该容器在 cgroup 下内存配置目录。

```
cd /sys/fs/cgroup/memory/docker/
```

```
[root@localhost ~]# cd /sys/fs/cgroup/memory/docker/
[root@localhost docker]# ll
total 0
drwxr-xr-x. 2 root root 0 Aug 12 22:10 507dd38c109086aa6a0c384bfbed722ff7e8e9fa1a0e9559ed4d2aee36268bbc
drwxr-xr-x. 2 root root 0 Aug 12 22:42 515da3dcef3ced12e714f66b3224f1f73588ab0b8eaa630166e8e6b955608911
drwxr-xr-x. 2 root root 0 Aug 12 22:25 5d88c92d5de9d601f77a69ebf219530d6127d78ba9fd73bb3c8dc1a62a52e9d2
drwxr-xr-x. 2 root root 0 Aug 12 22:25 b0a337e7370b203ec28cod7c3ad49c133d15a3dd8a2a81d4041af196f6f1b2a1
drwxr-xr-x. 2 root root 0 Aug 12 22:25 c46a346e516ec03f11786a639efafef50ef5c7ee642ac628dca7f788568df504
-rw-r--r--. 1 root root 0 Jul 10 09:14 cgroup.clone_children
--w--w--w-. 1 root root 0 Jul 10 09:14 cgroup.event_control
```

步骤 3 进入该容器的内存配置目录，查看内存配置文件。查看到内存使用限制为 400M，内存和 SWAP 资源为 500M。

```
cat memory.limit_in_bytes
```

```
[root@localhost docker]# cd 515da3dcef3ced12e714f66b3224f1f73588ab0b8eaa630166e8e6b955608911
[root@localhost 515da3dcef3ced12e714f66b3224f1f73588ab0b8eaa630166e8e6b955608911]# cat memory.limit_in_bytes
419430400
[root@localhost 515da3dcef3ced12e714f66b3224f1f73588ab0b8eaa630166e8e6b955608911]#
```

```
cat memory.memsw.limit_in_bytes
```

```
[root@localhost 515da3dcef3ced12e714f66b3224f1f73588ab0b8eaa630166e8e6b955608911]# cat memory.memsw.limit_in_bytes
524288000
```

步骤 4 为方便后续实验，删除本小节中的容器。

```
docker kill
docker rm
```