CPU资源分配实验及cgroups和systemd

模拟单核cpu系统

建立两个cpu类型的资源控制器，分别为richcpu CPU资源200，thincpu CPU资源为100，分别测试他们使用dd命令做读写相同大小的块文件的时间（其中cpu.shares最大不能超过1024）

使用lscpu 查看cpu的数量和核心数

```
[root@mysql ~]# lscpu | grep -i cpu
CPU op-mode(s):        32-bit, 64-bit
CPU(s):                2
On-line CPU(s) list:   0,1
CPU family:            6
Model name:            Intel(R) Core(TM) i7-10700KF CPU @ 3.80GHz
CPU MHz:               3792.002
NUMA node0 CPU(s):     0,1
```

为了保证实验公平掐灭第二颗cpu，cpu0为主cpu，不允许掐灭。使用echo "0" > /sys/devices/system/cpu/cpu1/online

```
[root@mysql cpu0]# echo "0" > /sys/devices/system/cpu/cpu1/online
[root@mysql cpu0]# lscpu |grep -i cpu
CPU op-mode(s):        32-bit, 64-bit
CPU(s):                2
On-line CPU(s) list:   0
Off-line CPU(s) list:  1
CPU family:            6
Model name:            Intel(R) Core(TM) i7-10700KF CPU @ 3.80GHz
CPU MHz:               3792.002
NUMA node0 CPU(s):     0
```

执行掐灭cpu1的命令后，再次使用lscpu |grep -i cpu，可以看到0号cpu为online状态，1号cpu为offline状态。

cgroup简介

cgroups(Control Groups) 是 linux 内核提供的一种机制（Linux 2.6.24内核开始将Cgroup加入主线），**这种机制可以根据需求把一系列系统任务及其子任务整合(或分隔)到按资源划分等级的不同组内，从而为系统资源管理提供一个统一的框架。**简单说，cgroups 主要用于限制和隔离一组进程对系统资源的使用，也就是做资源QoS。可控制的资源主要包括CPU、内存、block I/O、网络带宽等等。本质上来说，cgroups 是内核附加在程序上的一系列钩子(hook)，通过程序运行时对资源的调度触发相应的钩子以达到资源追踪和限制的目的。

cgroups 的主要作用

实现 cgroups 的主要目的是为不同用户层面的资源管理提供一个统一化的接口。从单个任务的资源控制到操作系统层面的虚拟化，cgroups 提供了四大功能：

- 资源限制：cgroups 可以对任务是要的资源总额进行限制。比如设定任务运行时使用的内存上限，一旦超出就发 OOM。

- 优先级分配：通过分配的 CPU 时间片数量和磁盘 IO 带宽，实际上就等同于控制了任务运行的优先级。
- 资源统计：cgoups 可以统计系统的资源使用量，比如 CPU 使用时长、内存用量等。这个功能非常适合当前云端产品按使用量计费的方式。
- 任务控制：cgroups 可以对任务执行挂起、恢复等操作。

cgroupfs

cgroups 以文件的方式提供应用接口，我们可以通过 mount 命令来查看 cgroups 默认的挂载点

```
[root@mysql cgroup]# mount |grep cgroup
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,mode=755)
cgroup on /sys/fs/cgroup/systemd type cgroup
(rw,nosuid,nodev,noexec,relatime,xattr,release_agent=/usr/lib/systemd/systemd-
cgroups-agent,name=systemd)
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup
(rw,nosuid,nodev,noexec,relatime,net_prio,net_cls)
cgroup on /sys/fs/cgroup/blkio type cgroup
(rw,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/hugetlb type cgroup
(rw,nosuid,nodev,noexec,relatime,hugetlb)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup
(rw,nosuid,nodev,noexec,relatime,cpuacct,cpu)
cgroup on /sys/fs/cgroup/devices type cgroup
(rw,nosuid,nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/cpuset type cgroup
(rw,nosuid,nodev,noexec,relatime,cpuset)
cgroup on /sys/fs/cgroup/perf_event type cgroup
(rw,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/freezer type cgroup
(rw,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/memory type cgroup
(rw,nosuid,nodev,noexec,relatime,memory)
```

Cgroup提供了一个原生接口并通过cgroupfs提供（从这句话我们可以知道cgroupfs就是Cgroup的一个接口的封装）。类似于procfs和sysfs，是一种虚拟文件系统。并且cgroupfs是可以挂载的，默认情况下挂载在/sys/fs/cgroup目录。如下图所示，我们可以看到对应的总资源管理及细节的资源管理：

```
[root@mysql cgroup]# pwd
/sys/fs/cgroup
[root@mysql cgroup]# ls -lt
total 0
lrwxrwxrwx 1 root root 11 Jan  4 17:47 cpu -> cpu,cpuacct
lrwxrwxrwx 1 root root 11 Jan  4 17:47 cpuacct -> cpu,cpuacct
drwxr-xr-x 2 root root  0 Jan  4 17:47 cpuset
drwxr-xr-x 4 root root  0 Jan  4 17:47 devices
drwxr-xr-x 2 root root  0 Jan  4 17:47 freezer
drwxr-xr-x 4 root root  0 Jan  4 17:47 memory
drwxr-xr-x 2 root root  0 Jan  4 17:47 perf_event
drwxr-xr-x 4 root root  0 Jan  4 17:47 blkio
drwxr-xr-x 4 root root  0 Jan  4 17:47 cpu,cpuacct
drwxr-xr-x 2 root root  0 Jan  4 17:47 hugetlb
lrwxrwxrwx 1 root root 16 Jan  4 17:47 net_cls -> net_cls,net_prio
drwxr-xr-x 2 root root  0 Jan  4 17:47 net_cls,net_prio
lrwxrwxrwx 1 root root 16 Jan  4 17:47 net_prio -> net_cls,net_prio
drwxr-xr-x 4 root root  0 Jan  4 17:47 pids
drwxr-xr-x 4 root root  0 Jan  4 17:47 systemd
[root@mysql cgroup]#
```

blkio 对块设备的 IO 进行限制。
cpu 限制 CPU 时间片的分配，与 cpuacct 挂载在同一目录。
cpuacct 生成 cgroup 中的任务占用 CPU 资源的报告，与 cpu 挂载在同一目录。
cpuset 给 cgroup 中的任务分配独立的 CPU(多处理器系统) 和内存节点。
devices 允许或禁止 cgroup 中的任务访问设备。
freezer 暂停/恢复 cgroup 中的任务。
hugetlb 限制使用的内存页数量。
memory 对 cgroup 中的任务的可用内存进行限制，并自动生成资源占用报告。
net_cls 使用等级识别符（classid）标记网络数据包，这让 Linux 流量控制器（tc 指令）可以识别来自特定 cgroup 任务的数据包，并进行网络限制。
net_prio 允许基于 cgroup 设置网络流量(netowork traffic)的优先级。
perf_event 允许使用 perf 工具来监控 cgroup。
pids 限制任务的数量。

```
[root@mysql cgroup]# cd cpuset/
[root@mysql cpuset]# ls -tl
total 0
-rw-r--r-- 1 root root 0 Jan  4 17:47 cgroup.clone_children
--w--w--w- 1 root root 0 Jan  4 17:47 cgroup.event_control
-rw-r--r-- 1 root root 0 Jan  4 17:47 cgroup.procs
-r--r--r-- 1 root root 0 Jan  4 17:47 cgroup.sane_behavior
-rw-r--r-- 1 root root 0 Jan  4 17:47 cpuset.cpu_exclusive
-rw-r--r-- 1 root root 0 Jan  4 17:47 cpuset.cpus
-r--r--r-- 1 root root 0 Jan  4 17:47 cpuset.effective_cpus
-r--r--r-- 1 root root 0 Jan  4 17:47 cpuset.effective_mems
-rw-r--r-- 1 root root 0 Jan  4 17:47 cpuset.mem_exclusive
-rw-r--r-- 1 root root 0 Jan  4 17:47 cpuset.mem_hardwall
-rw-r--r-- 1 root root 0 Jan  4 17:47 cpuset.memory_migrate
-r--r--r-- 1 root root 0 Jan  4 17:47 cpuset.memory_pressure
-rw-r--r-- 1 root root 0 Jan  4 17:47 cpuset.memory_pressure_enabled
-rw-r--r-- 1 root root 0 Jan  4 17:47 cpuset.memory_spread_page
-rw-r--r-- 1 root root 0 Jan  4 17:47 cpuset.memory_spread_slab
-rw-r--r-- 1 root root 0 Jan  4 17:47 cpuset.mems
-rw-r--r-- 1 root root 0 Jan  4 17:47 cpuset.sched_load_balance
-rw-r--r-- 1 root root 0 Jan  4 17:47 cpuset.sched_relax_domain_level
-rw-r--r-- 1 root root 0 Jan  4 17:47 notify_on_release
-rw-r--r-- 1 root root 0 Jan  4 17:47 release_agent
-rw-r--r-- 1 root root 0 Jan  4 17:47 tasks
```

可以看到这里有很多控制文件，其中以cpuset开头的控制文件都是cpuset子系统产生的，其他文件则由Cgroup产生。tasks文件记录了这个Cgroup的所有进程（包括线程），在系统启动后默认没有对Cgroup做任何配置的情况下，cgroupfs只有一个根目录

查看docker进程属于哪些cgroup

```
[root@mysql cgroup]# docker info |grep -i  Driver
WARNING: bridge-nf-call-iptables is disabled
WARNING: bridge-nf-call-ip6tables is disabled
 Storage Driver: overlay2
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
[root@mysql cgroup]# ps -ef|grep docker
root       1322      1  0 17:47 ?        00:00:01 /usr/bin/dockerd -H fd:// --
containerd=/run/containerd/containerd.sock
root       9798   4566  0 19:55 pts/2    00:00:00 grep --color=auto docker
[root@mysql cgroup]# more /proc/1322/cgroup
11:memory:/system.slice/docker.service
10:freezer:/
9:perf_event:/
8:cpuset:/
7:devices:/system.slice/docker.service
6:cpuacct,cpu:/system.slice/docker.service
5:hugetlb:/
4:blkio:/system.slice/docker.service
3:net_prio,net_cls:/
2:pids:/system.slice/docker.service
1:name=systemd:/system.slice/docker.service
```

cgroups工具

使用cgexec命令进行程序的测试

安装相关组件

```
yum install libcgroup libcgroup-tools -y
```

demo：限制进程可用的cpu

在我们使用 cgroups 时，最好不要直接在各个子系统的根目录下直接修改其配置文件。推荐的方式是为不同的需求在子系统树中定义不同的节点。比如我们可以在 /sys/fs/cgroup/cpu 目录下新建一个名称为 nick_cpu 的目录：

```
[root@mysql cpu]# pwd
/sys/fs/cgroup/cpu
[root@mysql cpu]# mkdir nick_cpu
[root@mysql cpu]# cd nick_cpu/
[root@mysql nick_cpu]# ls -tl
total 0
-rw-r--r-- 1 root root 0 Jan  5 09:26 cgroup.clone_children
--w--w--w- 1 root root 0 Jan  5 09:26 cgroup.event_control
-rw-r--r-- 1 root root 0 Jan  5 09:26 cgroup.procs
-r--r--r-- 1 root root 0 Jan  5 09:26 cpuacct.stat
-rw-r--r-- 1 root root 0 Jan  5 09:26 cpuacct.usage
-r--r--r-- 1 root root 0 Jan  5 09:26 cpuacct.usage_percpu
-rw-r--r-- 1 root root 0 Jan  5 09:26 cpu.cfs_period_us
-rw-r--r-- 1 root root 0 Jan  5 09:26 cpu.cfs_quota_us
-rw-r--r-- 1 root root 0 Jan  5 09:26 cpu.rt_period_us
-rw-r--r-- 1 root root 0 Jan  5 09:26 cpu.rt_runtime_us
-rw-r--r-- 1 root root 0 Jan  5 09:26 cpu.shares
-r--r--r-- 1 root root 0 Jan  5 09:26 cpu.stat
-rw-r--r-- 1 root root 0 Jan  5 09:26 notify_on_release
-rw-r--r-- 1 root root 0 Jan  5 09:26 tasks
```

通过下面的设置把 CPU 周期限制为总量的十分之一

```
echo 100000 > nick_cpu/cpu.cfs_period_us
echo 10000 > nick_cpu/cpu.cfs_quota_us
```

创建一个 CPU 密集型的程序

```
#include <stdio.h>
#include <stdlib.h>
void main(){
    unsigned int i, end;

    end = 1024 * 1024 * 1024;
    for(i = 0; i < end; )
    {
        i ++;
    }
}
```

编译程序后执行程序

```
[root@mysql cprogram]# time ./test01
real    0m1.488s
user    0m1.484s
sys 0m0.001s
[root@mysql cprogram]# time cgexec -g cpu:nick_cpu ./test01
real    0m20.600s
user    0m2.062s
sys 0m0.001s
[root@mysql cprogram]#
```

time 命令可以为我们报告程序执行消耗的时间，其中的 real 就是我们真实感受到的时间。使用 cgexec 能够把我们添加的 cgroup 配置 nick_cpu 应用到运行 cputime 程序的进程上。 上图显示，默认的执行只需要 2s 左右。通过 cgroups 限制 CPU 资源后需要运行 23s。

demo：限制进程可用的内存

限制进程可用的最大内存，在 /sys/fs/cgroup/memory 下创建目录nick_memory

```
[root@mysql memory]# mkdir nick_memory
[root@mysql memory]# cd nick_memory/
[root@mysql nick_memory]# ls -tl
total 0
-rw-r--r-- 1 root root 0 Jan  5 09:35 cgroup.clone_children
--w--w--w- 1 root root 0 Jan  5 09:35 cgroup.event_control
-rw-r--r-- 1 root root 0 Jan  5 09:35 cgroup.procs
-rw-r--r-- 1 root root 0 Jan  5 09:35 memory.failcnt
--w------- 1 root root 0 Jan  5 09:35 memory.force_empty
-rw-r--r-- 1 root root 0 Jan  5 09:35 memory.kmem.failcnt
-rw-r--r-- 1 root root 0 Jan  5 09:35 memory.kmem.limit_in_bytes
-rw-r--r-- 1 root root 0 Jan  5 09:35 memory.kmem.max_usage_in_bytes
-r--r--r-- 1 root root 0 Jan  5 09:35 memory.kmem.slabinfo
-rw-r--r-- 1 root root 0 Jan  5 09:35 memory.kmem.tcp.failcnt
-rw-r--r-- 1 root root 0 Jan  5 09:35 memory.kmem.tcp.limit_in_bytes
-rw-r--r-- 1 root root 0 Jan  5 09:35 memory.kmem.tcp.max_usage_in_bytes
-r--r--r-- 1 root root 0 Jan  5 09:35 memory.kmem.tcp.usage_in_bytes
-r--r--r-- 1 root root 0 Jan  5 09:35 memory.kmem.usage_in_bytes
-rw-r--r-- 1 root root 0 Jan  5 09:35 memory.limit_in_bytes
-rw-r--r-- 1 root root 0 Jan  5 09:35 memory.max_usage_in_bytes
-rw-r--r-- 1 root root 0 Jan  5 09:35 memory.memsw.failcnt
-rw-r--r-- 1 root root 0 Jan  5 09:35 memory.memsw.limit_in_bytes
-rw-r--r-- 1 root root 0 Jan  5 09:35 memory.memsw.max_usage_in_bytes
-r--r--r-- 1 root root 0 Jan  5 09:35 memory.memsw.usage_in_bytes
-rw-r--r-- 1 root root 0 Jan  5 09:35 memory.move_charge_at_immigrate
-r--r--r-- 1 root root 0 Jan  5 09:35 memory.numa_stat
-rw-r--r-- 1 root root 0 Jan  5 09:35 memory.oom_control
---------- 1 root root 0 Jan  5 09:35 memory.pressure_level
-rw-r--r-- 1 root root 0 Jan  5 09:35 memory.soft_limit_in_bytes
-r--r--r-- 1 root root 0 Jan  5 09:35 memory.stat
-rw-r--r-- 1 root root 0 Jan  5 09:35 memory.swappiness
-r--r--r-- 1 root root 0 Jan  5 09:35 memory.usage_in_bytes
-rw-r--r-- 1 root root 0 Jan  5 09:35 memory.use_hierarchy
-rw-r--r-- 1 root root 0 Jan  5 09:35 notify_on_release
-rw-r--r-- 1 root root 0 Jan  5 09:35 tasks
```

设置进程的可用内存限制在最大 300M，并且不使用 swap

```
[root@mysql memory]# echo 314572800 > nick_memory/memory.limit_in_bytes
[root@mysql memory]# echo 0 > nick_memory/memory.swappiness
```

创建一个不断分配内存的程序，它分五次分配内存，每次申请 100M,编译程序后执行

```
[root@mysql cprogram]# ./memory
malloc memory 100 MB
malloc memory 200 MB
malloc memory 300 MB
malloc memory 400 MB
malloc memory 500 MB
[root@mysql cprogram]# cgexec -g memory:nick_memory ./memory
malloc memory 100 MB
malloc memory 200 MB
Killed
[root@mysql cprogram]#
```

由于内存不足且禁止使用 swap，所以被限制资源的进程在申请内存时被强制杀死了。

实际应用中往往要同时限制多种的资源，比如既限制 CPU 资源又限制内存资源。使用 cgexec 实现这样的用例其实很简单，直接指定多个 -g 选项就可以

```
cgexec -g memory:nick_memory -g cpu:nick_cpu ./memory
```

systemd

当 Linux 的 init 系统发展到 systemd 之后，systemd 与 cgroups 发生了融合(或者说 systemd 提供了 cgroups 的使用和管理接口，systemd 管的东西越来越多啊！)。Systemd也是对于Cgroup接口的一个封装。systemd以PID1的形式在系统启动的时候运行，并提供了一套系统管理守护程序、库和实用程序，用来控制、管理Linux计算机操作系统资源。通过**systemd-cgls**命令我们可以看到systemd工作的进程PID是1，而目录/sys/fs/cgroup/systemd是systemd维护的自己使用的非subsystem的cgroups层级结构。

Systemd限制CPU、内存的简单操作方法

> systemctl set-property cron.service CPUShares=100 MemoryLimit=200M

Systemd 依赖 cgroups

要理解 systemd 与 cgroups 的关系，我们需要先区分 cgroups 的两个方面：**层级结构(A)和资源控制(B)**。首先 cgroups 是以层级结构组织并标识进程的一种方式，同时它也是在该层级结构上执行资源限制的一种方式。我们简单的把 cgroups 的层级结构称为 A，把 cgrpups 的资源控制能力称为 B。
对于 systemd 来说，A 是必须的，如果没有 A，systemd 将不能很好的工作。而 B 则是可选的，如果你不需要对资源进行控制，那么在编译 Linux 内核时完全可以去掉 B 相关的编译选项。

Systemd 默认挂载的 cgroups 系统

在系统的开机阶段，systemd 会把支持的 controllers (subsystem 子系统)挂载到默认的 /sys/fs/cgroup/ 目录下面

```
[root@mysql cgroup]# ls -tl
total 0
lrwxrwxrwx 1 root root 11 Jan  4 17:47 cpu -> cpu,cpuacct
```

```
lrwxrwxrwx 1 root root 11 Jan  4 17:47 cpuacct -> cpu,cpuacct
drwxr-xr-x 2 root root  0 Jan  4 17:47 cpuset
drwxr-xr-x 4 root root  0 Jan  4 17:47 devices
drwxr-xr-x 2 root root  0 Jan  4 17:47 freezer
drwxr-xr-x 5 root root  0 Jan  4 17:47 memory
drwxr-xr-x 2 root root  0 Jan  4 17:47 perf_event
drwxr-xr-x 4 root root  0 Jan  4 17:47 blkio
drwxr-xr-x 5 root root  0 Jan  4 17:47 cpu,cpuacct
drwxr-xr-x 2 root root  0 Jan  4 17:47 hugetlb
lrwxrwxrwx 1 root root 16 Jan  4 17:47 net_cls -> net_cls,net_prio
drwxr-xr-x 2 root root  0 Jan  4 17:47 net_cls,net_prio
lrwxrwxrwx 1 root root 16 Jan  4 17:47 net_prio -> net_cls,net_prio
drwxr-xr-x 4 root root  0 Jan  4 17:47 pids
drwxr-xr-x 4 root root  0 Jan  4 17:47 systemd
```

/sys/fs/cgroup/systemd 目录是 systemd 维护的自己使用的非 subsystem 的 cgroups 层级结构。这玩意儿是 systemd 自己使用的，换句话说就是，并不允许其它的程序动这个目录下的内容。其实 /sys/fs/cgroup/systemd 目录对应的 cgroups 层级结构就是 systemd 用来使用 cgoups 中 feature A 的。

Cgroup 的默认层级

通过将 cgroup 层级系统与 systemd unit 树绑定，systemd 可以把资源管理的设置从进程级别移至应用程序级别。因此，我们可以使用 systemctl 指令，或者通过修改 systemd unit 的配置文件来管理 unit 相关的资源。

默认情况下，systemd 会自动创建 **slice、scope 和 service** unit 的层级(slice、scope 和 service 都是 systemd 的 unit 类型，参考《初识 systemd》)，来为 cgroup 树提供统一的层级结构。

系统中运行的所有进程，都是 systemd init 进程的子进程。在资源管控方面，systemd 提供了三种 unit 类型：

- **service**： 一个或一组进程，由 systemd 依据 unit 配置文件启动。service 对指定进程进行封装，这样进程可以作为一个整体被启动或终止。
- **scope**：一组外部创建的进程。由进程通过 fork() 函数启动和终止、之后被 systemd 在运行时注册的进程，scope 会将其封装。例如：用户会话、 容器和虚拟机被认为是 scope。
- **slice**： 一组按层级排列的 unit。slice 并不包含进程，但会组建一个层级，并将 scope 和 service 都放置其中。真正的进程包含在 scope 或 service 中。在这一被划分层级的树中，每一个 slice 单位的名字对应通向层级中一个位置的路径。

可以使用systemd-cgls 查看cgroup层级关系

```
[root@mysql ~]# systemd-cgls
├─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 22
├─user.slice
| └─user-0.slice
|   ├─session-6.scope
|   | ├─4564 sshd: root@pts/2
|   | └─4566 -bash
|   ├─session-5.scope
|   | ├─4239 sshd: root@pts/1
|   | └─4241 -bash
|   ├─session-3.scope
|   | ├─ 3406 sshd: root@pts/0
|   | ├─ 3410 -bash
|   | ├─53636 systemd-cgls
|   | └─53637 systemd-cgls
```

```
|   └─session-1.scope
|     ├─ 832 login -- root
|     └─2477 -bash
└─system.slice
  ├─docker.service
  | └─1322 /usr/bin/dockerd -H fd:// --
containerd=/run/containerd/containerd.sock
  ├─rhsmcertd.service
  | └─1419 /usr/bin/rhsmcertd
  ├─sshd.service
  | └─1295 /usr/sbin/sshd -D
  ├─postgresql-9.6.service
```

service、scope 和 slice unit 被直接映射到 cgroup 树中的对象。当这些 unit 被激活时，它们会直接一一映射到由 unit 名建立的 cgroup 路径中。例如，cron.service 属于 system.slice，会直接映射到 cgroup system.slice/cron.service/ 中。
注意，所有的用户会话、虚拟机和容器进程会被自动放置在一个单独的 scope 单元中。

默认情况下，系统会创建四种 slice：

- **-.slice**：根 slice
- **system.slice**：所有系统 service 的默认位置
- **user.slice**：所有用户会话的默认位置
- **machine.slice**：所有虚拟机和 Linux 容器的默认位置

创建临时的cgroup

对资源管理的设置可以是 transient(临时的)，也可以是 persistent (永久的)。我们先来介绍如何创建临时的 cgroup。
需要使用 **systemd-run** 命令创建临时的 cgroup，它可以创建并启动临时的 service 或 scope unit，并在此 unit 中运行程序。systemd-run 命令默认创建 service 类型的 unit，比如我们创建名称为 toptest 的 service 运行 top 命令

```
[root@mysql ~]# systemd-run --unit=toptest --slice=test top -b
Running as unit toptest.service.
[root@mysql ~]# systemctl status toptest
● toptest.service - /usr/bin/top -b
   Loaded: loaded (/run/systemd/system/toptest.service; static; vendor preset:
disabled)
  Drop-In: /run/systemd/system/toptest.service.d
           └─50-Description.conf, 50-ExecStart.conf, 50-Slice.conf
   Active: active (running) since Tue 2021-01-05 09:56:16 CST; 7s ago
 Main PID: 53795 (top)
   CGroup: /test.slice/toptest.service
           └─53795 /usr/bin/top -b

Jan 05 09:56:22 mysql top[53795]: 3672 root      20   0       0      0      0 S
0.0  0.0   0:00.00 kworker/1:1
Jan 05 09:56:22 mysql top[53795]: 4239 root      20   0  159344   5960   4592 S
0.0  0.2   0:00.16 sshd
Jan 05 09:56:22 mysql top[53795]: 4241 root      20   0  116856   3580   1788 S
0.0  0.1   0:00.06 bash
Jan 05 09:56:22 mysql top[53795]: 4564 root      20   0  159344   5964   4592 S
0.0  0.2   0:00.46 sshd
Jan 05 09:56:22 mysql top[53795]: 4566 root      20   0  116860   3684   1840 S
0.0  0.1   0:00.20 bash
```

```
Jan 05 09:56:22 mysql top[53795]: 50026 postfix   20   0   91840   4080   3068 S
0.0  0.1   0:00.00 pickup
Jan 05 09:56:22 mysql top[53795]: 52349 root      20   0       0      0      0 S
0.0  0.0   0:01.23 kworker/0:0
Jan 05 09:56:22 mysql top[53795]: 52432 root      20   0       0      0      0 S
0.0  0.0   0:00.02 kworker/u2+
Jan 05 09:56:22 mysql top[53795]: 53434 root      20   0       0      0      0 S
0.0  0.0   0:00.00 kworker/0:2
Jan 05 09:56:22 mysql top[53795]: 53700 root      20   0       0      0      0 S
0.0  0.0   0:00.12 kworker/0:1
```

下面可以通过 systemctl 命令来限制 toptest.service 的资源了，首先查看当前toptest.service的cgroup资源限制

```
[root@mysql ~]# more /proc/53795/cgroup
11:memory:/test.slice
10:freezer:/
9:perf_event:/
8:cpuset:/
7:devices:/test.slice
6:cpuacct,cpu:/test.slice
5:hugetlb:/
4:blkio:/test.slice
3:net_prio,net_cls:/
2:pids:/test.slice
1:name=systemd:/test.slice/toptest.service
```

限制 toptest.service 的 CPUShares 为 600，可用内存的上限为 550M

```
[root@mysql ~]# systemctl set-property toptest.service CPUShares=600
MemoryLimit=500M
[root@mysql ~]# more /proc/53795/cgroup
11:memory:/test.slice/toptest.service
10:freezer:/
9:perf_event:/
8:cpuset:/
7:devices:/test.slice
6:cpuacct,cpu:/test.slice/toptest.service
5:hugetlb:/
4:blkio:/test.slice
3:net_prio,net_cls:/
2:pids:/test.slice
1:name=systemd:/test.slice/toptest.service
```

在 CPU 和 memory 子系统中都出现了 toptest.service 的名字。同时去查看 **/sys/fs/cgroup/memory/test.slice** 和 **/sys/fs/cgroup/cpu/test.slice** 目录，这两个目录下都多出了一个 toptest.service 目录。我们设置的 CPUShares=600 MemoryLimit=500M 被分别写入了这些目录下的对应文件中。

```
[root@mysql toptest.service]# pwd
/sys/fs/cgroup/cpu/test.slice/toptest.service
[root@mysql toptest.service]# more cpu.shares
600
```

临时 cgroup 的特征是，所包含的进程一旦结束，临时 cgroup 就会被自动释放。比如我们 kill 掉 top 进程，然后再查看 /sys/fs/cgroup/memory/test.slice 和 /sys/fs/cgroup/cpu/test.slice 目录，刚才的 toptest.service 目录已经不见了。

配置文件修改 cgroup

systemd 监管的 persistent cgroup(持久的 cgroup)都在 /usr/lib/systemd/system/ 目录中有一个 unit 配置文件。比如我们常见的 service 类型 unit 的配置文件。我们可以通过设置 unit 配置文件来控制应用程序的资源，persistent cgroup 的特点是即便系统重启，相关配置也会被保留

编辑/usr/lib/systemd/system/xxxx配置文件，添加相关的cpu资源限制或内存限制等

Systemd-cgtop 命令

类似于 top 命令，systemd-cgtop 命令显示 cgoups 的实时资源消耗情况：

```
[root@mysql system]# systemd-cgtop
Path                                                                  Tasks   %CPU   Memory  Input/s Output/s

/                                                                        78      -     1.1G        -        -
/system.slice                                                             -      -   696.2M        -        -
/system.slice/NetworkManager.service                                      1      -    11.5M        -        -
/system.slice/auditd.service                                              1      -     2.6M        -        -
/system.slice/chronyd.service                                             1      -     1.4M        -        -
/system.slice/containerd.service                                          1      -    64.0M        -        -
/system.slice/crond.service                                               1      -   628.0K        -        -
/system.slice/dbus.service                                                1      -     1.6M        -        -
```

refer:https://www.cnblogs.com/sparkdev/p/8296063.html

https://www.cnblogs.com/sparkdev/p/9523194.html

https://www.cnblogs.com/xiaoyaojinzhazhadehangcheng/p/11606500.html