Cgroups 与 Systemd

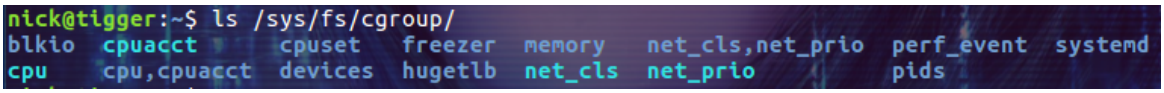
Cgroups 是 linux 内核提供的一种机制，如果你还不了解 cgroups，请参考前文《Linux cgroups 简介》先了解 cgroups。当 Linux 的 init 系统发展到 systemd 之后，systemd 与 cgroups 发生了融合(或者说 systemd 提供了 cgroups 的使用和管理接口，systemd 管的东西越来越多啊！)。本文将简单的介绍 cgroups 与 systemd 的关系以及如何通过 systemd 来配置和使用 cgroups。

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：



除了 systemd 目录外，其它目录都是对应的 subsystem。
/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。

我们可以通过 systemd-cgls 命令来查看 cgroups 的层级结构：

```
Control group /:
-.slice
├─user.slice
│ └─user-1000.slice
│   ├─user@1000.service
│   │ └─init.scope
│   │   ├─1104 /lib/systemd/systemd --user
│   │   └─1105 (sd-pam)
│   └─session-c1.scope
│     ├─1099 lightdm --session-child 12 15
│     ├─1108 /sbin/upstart --user
│     ├─1185 /bin/sh -e /proc/self/fd/9
│     ...
├─init.scope
│ └─1 /sbin/init splash
└─system.slice
  ├─irqbalance.service
  │ └─1000 /usr/sbin/irqbalance --pid=/var/run/irqbalance.pid
  ├─lvm2-lvmetad.service
  │ └─419 /sbin/lvmetad -f
  ├─cron.service
  │ └─915 /usr/sbin/cron -f
  ...
```

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

```
Control group /:
-.slice
├─user.slice
│ └─user-1000.slice
│   ├─user@1000.service
│   │ └─init.scope
│   │   ├─1104 /lib/systemd/systemd --user
│   │   └─1105 (sd-pam)
│   └─session-c1.scope
│     ├─1099 lightdm --session-child 12 15
│     ├─1108 /sbin/upstart --user
│     ├─1185 /bin/sh -e /proc/self/fd/9
│     ...
│     └─2110 systemd-cgls
├─init.scope
│ └─1 /sbin/init splash
└─system.slice
  ├─irqbalance.service
  │ └─1000 /usr/sbin/irqbalance --pid=/var/run/irqbalance.pid
  ├─lvm2-lvmetad.service
  │ └─419 /sbin/lvmetad -f
  ...
  ├─systemd-hostnamed.service
  │ └─1065 /lib/systemd/systemd-hostnamed
  └─cups-browsed.service
    └─1018 /usr/sbin/cups-browsed
lines 99-153/153 (END)
```

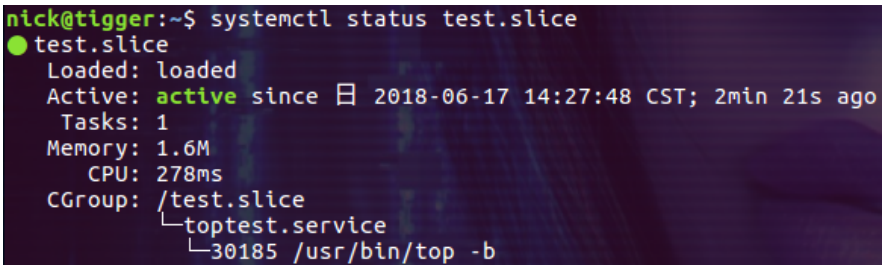
创建临时的 cgroup

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：

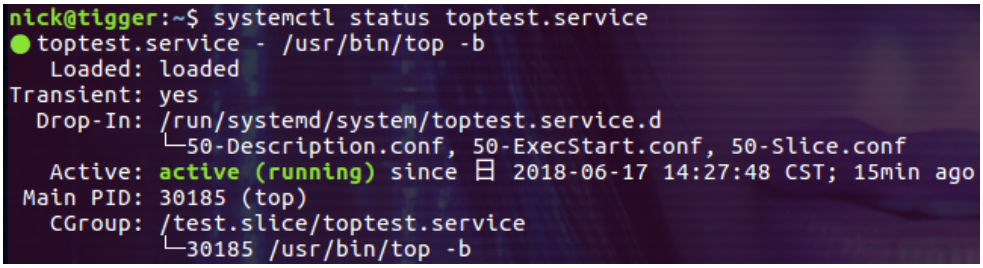
```
$ sudo systemd-run --unit=toptest --slice=test top -b
```

```
nick@tigger:~$ sudo systemd-run --unit=toptest --slice=test top -b
Running as unit toptest.service.
```

然后查看一下 test.slice 的状态：



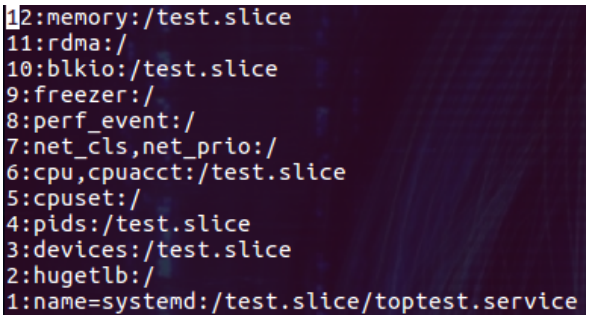
创建了一个 test.slice/toptest.service cgroup 层级关系。再看看 toptest.service 的状态：



top 命令被包装成一个 service 运行在后台了！

接下来我们就可以通过 systemctl 命令来限制 toptest.service 的资源了。在限制前让我们先来看一看 top 进程的 cgroup 信息：

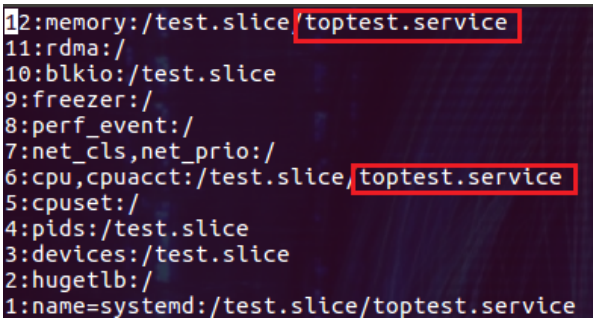
```
$ vim /proc/2850/cgroup          # 2850 为 top 进程的 PID
```



比如我们限制 toptest.service 的 CPUShares 为 600，可用内存的上限为 550M：

```
$ sudo systemctl set-property toptest.service CPUShares=600 MemoryLimit=500M
```

再次检查 top 进程的 cgroup 信息：



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。

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。

通过配置文件修改 cgroup

所有被 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。需要注意的是，scope unit 不能以此方式创建。下面让我们为 cron.service 添加 CPU 和内存相关的一些限制，编辑 /lib/systemd/system/cron.service 文件：

```
$ sudo vim  /lib/systemd/system/cron.service
```

```
[Service]
CPUShares=600
MemoryLimit=500M
EnvironmentFile=-/etc/default/cron
ExecStart=/usr/sbin/cron -f $EXTRA_OPTS
IgnoreSIGPIPE=false
KillMode=process
```

添加红框中的行，然后重新加载配置文件并重启 cron.service：

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart cron.service
```

现在去查看 /sys/fs/cgroup/memory/system.slice/cron.service/memory.limit_in_bytes 和 /sys/fs/cgroup/cpu/system.slice/cron.service/cpu.shares 文件，是不是已经包含我们配置的内容了！

通过 systemctl 命令修改 cgroup

除了编辑 unit 的配置文件，还可以通过 systemctl set-property 命令来修改 cgroup，这种方式修该的配置也会在重启系统时保存下来。现在我们把 cron.service 的 CPUShares 改为 700：

```
$ sudo systemctl set-property cron.service CPUShares=700
```

查看 /sys/fs/cgroup/cpu/system.slice/cron.service/cpu.shares 文件的内容应该是 700，重启系统后该文件的内容还是 700。

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：

```
Control Group                              Tasks   %CPU   Memory  Input/s Output/s
/                                              -    2.4  1004.5M        -        -
/user.slice                                  219    1.6   590.2M        -        -
/system.slice                                 93    0.7   480.0M        -        -
/system.slice/lightdm.service                  5    0.5   103.5M        -        -
/system.slice/docker.service                  19    0.2   120.2M        -        -
/system.slice/open-vm-tools.service            1    0.0    15.7M        -        -
/system.slice/accounts-daemon.service          3    0.0     7.5M        -        -
/init.scope                                    1      -     8.1M        -        -
```

通过它我们就可以分析应用使用资源的情况。

总结

Systemd 是一个强大的 init 系统，它甚至为我们使用 cgorups 提供了便利！Systemd 提供的内在机制、默认设置和相关的操控命令降低了配置和使用 cgroups 的难度，即便是 Linux 新手，也能轻松的使用 cgroups 了。

参考：

The New Control Group Interfaces
systemd for Administrators, Part XVIII
Control Groups vs. Control Groups
RedHat Cgroups doc
Systemd-cgls
Systemd-cgtop