本文讨论如何使用cgroup限制Linux用户各种资源使用，主要演示了CPU和内存的限制方法。

注意：因Linux发行版本的差异，本方法并不适用于所有Linux发行版。以下方法在Redhat 7/CentOS 7上测试通过。

一、目的

限制Linux特定用户（的所有进程）资源使用上限，例如：限制用户`yz`的CPU使用上限为10%。

二、准备

2.1 重要说明

- 文中的`cgroup`配置需要Linux root权限。
- 所述操作需要在集群的所有物理节点上执行。

2.2 安装libcgroup

```
[root@bogon ~]# yum install libcgroup
[root@bogon ~]# yum install libcgroup-tools
```

2.3 cgroup子系统

子系统（`sub system`）是`cgroup`的内核子模块，被看作资源控制器，为不同`cgroup`配不同级别的系统资源。这里仅列出我们用到的子系统（其它子系统可参阅文献1）：

- `cpu`：CPU资源控制器，可以控制CPU使用率与调度策略等。
- `cpuset`：CPU核控制器，可以控制资源组只使用哪些CPU核。
- `puacct`：CPU统计子系统，用于统计与报告CPU使用情况，例如资源组实际的CPU使用时间等。
- `memory`：内存控制与统计子系统，可以控制内存使用量与使用策略，也提供内存使用情况报告。

三、配置

一句话概括就是，先创建资源组（描述各种资源的限制值，也可称为`cgroup`），然后建立用户与资源组之间的关联规则（`rule`），即哪个用户用哪个资源组限制什么资源。

3.1 配置方法

有两个相关的系统配置文件：

- `/etc/cgconfig.conf`：`cgroup`资源组描述信息文件，配置每个资源组各种资源（CPU、内存、IO等）的配额。为了不影响`/etc/cgconfig.conf`文件，我们还可以将配置文件放到`/etc/cgconfig.d/`目录下（如果目录不存在，可以手动创建。）。
- `/etc/cgrules.conf`：规则配置文件，将Linux用户与`/etc/cgconfig.conf`中的资源组进行配对，即哪个用户用哪个资源组限制什么资源。

3.2 例子

【例子】：假设要限制`Linux`用户`seabox`的资源：

- CPU最多使用一个核的`80%`（即使有多个核，也只能用一个核的`80%`）。
- 物理内存使用上限为`10GB`，总内存（物理和`swap`）使用上限为`15GB`。

首先，编写`cgconfig`配置文件`/etc/cgconfig.d/seaboxdb.conf`（注：`seaboxdb.conf`是我们自己取的名字，扩展名必须是`.conf`。）

```
group seabox_limit {
  cpu {
    cpu.cfs_quota_us = 80000;
    cpu.cfs_period_us = 100000;
  }
  memory {
    memory.limit_in_bytes = 10G;
    memory.memsw.limit_in_bytes = 15G;
  }
}
```

然后，在`/etc/cgrules.conf`中添加配置，将`seabox`用户关联到`seabox_limit`资源组。

```
seabox          cpu,memory          seabox_limit/
```

这行配置意思是：`seabox`用户的CPU和内存使用`seabox_limit`资源组限制。

最后，需要重启`cgroup`相关服务：

```
[root@bogon ~]# systemctl enable cgconfig    # 启用cgconfig
[root@bogon ~]# systemctl restart cgconfig   # 重启cgconfig，将cgconfig配置文件中的配置写入cgroup
[root@bogon ~]# systemctl enable cgred       # 启用cgred
[root@bogon ~]# systemctl restart cgred      # 重启cgred，将规则配置文件的配置写入cgroup
```

注意：

- 以上两个`.conf`文件不得被删除，因为系统重启时，`cgconfig`服务仍然需要读取这些配置文件以重建资源组。
- 如果修改了`cgroup`配置，则需要重启`cgroup`相关服务。
- `cgroup`组名称不能重复，也不能与已有的组名称重复。

3.3 复杂配置范例

```
group seabox_limit {
  cpu {
    # 最多使用4个CPU核的80%：100000 * 4 * 0.8 = 320000
    cpu.cfs_quota_us = 320000;
    cpu.cfs_period_us = 100000;
  }
  cpuacct {
  }
  cpuset {
    # 设置可用内存节点，通常设置为全部可用内存节点即可。
    cpuset.mems = '0-1';
    # 仅使用0、1、2和3这4个CPU核
    cpuset.cpus = '0-3';
  }
  memory {
    # 最多使用50GB物理内存
    memory.limit_in_bytes = 50G;
    # 最多使用60GB物理和swap内存
    memory.memsw.limit_in_bytes = 60G;
  }
}
```

获得全部可用内存节点的方法：

方法1：查询cpuset子系统根节点的`cpuset.mems`

```
[root@bogon ~]# cat /sys/fs/cgroup/cpuset/cpuset.mems
0-1
```

方法2：用`numactl`查询

```
[root@bogon ~]# numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 2
node 0 size: 9215 MB
node 0 free: 283 MB
node 1 cpus: 1 3
node 1 size: 9215 MB
node 1 free: 321 MB
node distances:
node   0    1
  0:  10   21
  1:  21   10
```

3.4 校验方法

本节说明如何检查以上配置是否写入`cgroup`系统。

Step 1. 确定子系统挂载点（`cgroup`子系统在文件系统中的虚拟文件）。

```
[root@bogon ~]# lssubsys -am
cpuset /sys/fs/cgroup/cpuset
cpu,cpuacct /sys/fs/cgroup/cpu,cpuacct
memory /sys/fs/cgroup/memory
devices /sys/fs/cgroup/devices
freezer /sys/fs/cgroup/freezer
net_cls,net_prio /sys/fs/cgroup/net_cls,net_prio
blkio /sys/fs/cgroup/blkio
perf_event /sys/fs/cgroup/perf_event
hugetlb /sys/fs/cgroup/hugetlb
pids /sys/fs/cgroup/pids
```

从以上输出可以知道：`cpuset`子系统挂载在`/sys/fs/cgroup/cpuset`，`cpu`和`cpuacct`子系统挂载在`/sys/fs/cgroup/cpu,cpuacct`，……

Step 2. 检查子系统下是否创建了`cgroup`组。

以`cpu`子系统下的`cgroup`组为例，其它子系统类似。

```
[root@bogon ~]# ls -al /sys/fs/cgroup/cpu,cpuacct/
drwxr-xr-x   9 root root   0 3月   8 11:26 .
drwxr-xr-x  13 root root 340 2月  13 11:36 ..
-rw-r--r--   1 root root   0 2月  13 11:36 cgroup.clone_children
……
drwxr-xr-x   2 root root   0 3月   8 11:16 seabox_limit
……
```

可以看到`cpu`子系统下已经创建了`seabox_limit`组。

Step 3. `cgroup`组下的参数值是否正确。

以`seabox_limit`组的`cpu.cfs_quota_us`参数为例，其它参数类似。

```
[root@bogon ~]# cat /sys/fs/cgroup/cpu,cpuacct/seabox_limit/cpu.cfs_quota_us
80000
```

也可以用以下命令：

```
[root@bogon ~]# cgget -g cpu /seabox_limit/
/seabox_limit/:
cpu.rt_period_us: 1000000
cpu.rt_runtime_us: 0
cpu.stat: nr_periods 7257
        nr_throttled 2912
        throttled_time 43156965570
cpu.cfs_period_us: 100000
cpu.cfs_quota_us: 80000
cpu.shares: 1024
```

四、多用户配置

多用户是以上单个用户的简单扩展。

`cgconfig`配置文件类似这样：

```
group user1_limit {
  cpu {
    ……
  }
  cpuset {
    ……
  }
  cpuacct {
  }
  memory {
    ……
  }
}

group user2_limit {
  cpu {
    ……
  }
  memory {
    ……
  }
}

……
```

而`cgrules`配置类似这样：

```
user1        cpu,cpuset,cpuacct,memory        user1_limit/
user2        cpu,memory                       user2_limit/
……
```

五、资源限制参数说明

本章按照子系统分类说明部分常用参数，完整参数说明见cgroup用户手册。

5.1 cpu

cpu.cfs_period_us

设定CPU的时间片周期，是一个基准值，单位为微秒（μs，这里以 “us” 表示）。与cpu.cfs_quota_us配套使用。

通常设置为100000。上限为 1 秒，下限为 1000 微秒。

cpu.cfs_quota_us

设定在某一周期（由 `cpu.cfs_period_us` 规定）某个 `cgroup` 所有任务可运行的时间总量上限，单位为微秒。一旦 cgroup 中任务用完配额时间，它们就会被限制流量，并在进入下阶段前禁止运行。这样就限制了该 cgroup 的CPU使用率。

`cpu.cfs_quota_us` 为 -1，表示该 cgroup 的 CPU 使用时间不受限制。

【例子】：假设系统总共有4个CPU核，欲限制某个 cgroup 最多使用全部CPU资源的50%，则配置如下：

```
cpu.cfs_period_us = 100000;
cpu.cfs_quota_us = 200000;
```

解释一下，每个CPU核时间周期是100000微妙，那么 `cpu.cfs_quota_us` 设置为200000毫秒，表示可以使用2倍的单核CPU时间，即可以使用全部CPU核的50%资源。

5.2 cpuset

cpuset.cpus

设定该 cgroup 任务可以使用的 CPU核。这是一个逗号分隔列表，格式为 ASCII，小横线（"-"）代表范围。例如：

```
0-3,16
```

表示 CPU核：0、1、2、3 和 16。

cpuset.mems

设定该 cgroup 中任务可以使用的内存节点。格式同 `cpuset.cpus`。

注意：设置cpuset.cpus必须同时设置 `cpuset.mems`。

5.3 cpuacct

cpuacct子系统作用是统计和报告CPU使用情况，如果不关心可以不配置。

cpuacct.usage

报告此 cgroup 中所有任务（包括层级中的低端任务）使用 CPU 的总时间（纳秒）。

cpuacct.stat

报告此 cgroup 的所有任务（包括层级中的低端任务）使用的用户和系统 CPU 时间。

cpuacct.usage_percpu

报告 cgroup 中所有任务（包括层级中的低端任务）在每个 CPU 中使用的 CPU 时间（纳秒）。

5.4 memory

memory子系统参数分为两大类：控制类与统计报告类。

控制类包括：

memory.limit_in_bytes

设定物理内存的最大使用量。如果没有指定单位，则默认为字节。但是可以使用后缀代表更大的单位（k 或者 K 代表千字节，m 或者 M 代表兆字节，g 或者 G 代表千兆字节）。

`memory.limit_in_bytes` 设为 -1，表示不限制。

memory.memsw.limit_in_bytes

设定物理内存与 swap 用量之和的最大值。单位同`memory.limit_in_bytes`。

`memory.memsw.limit_in_bytes` 设为 -1，表示不限制。

注意：`memory.limit_in_bytes`必须在 `memory.memsw.limit_in_bytes` 参数之前设定，顺序颠倒会导致错误。

【例子】：为某一 cgroup 设定 `memory.limit_in_bytes = 2G` 和 `memory.memsw.limit_in_bytes = 4G`，可以让该 cgroup 中的进程最多分得 2GB 物理内存，并且一旦用尽，只能再分得 2GB swap。`memory.memsw.limit_in_bytes` 参数表示物理内存和 swap 的总和。没有设置 `memory.memsw.limit_in_bytes` 参数的 cgroup 的进程可以使用全部可用 swap （当限定的内存用尽后），并会因为缺少可用 swap 触发 OOM 状态。

如果使用`cgconfig.conf`配置，也需要遵照此顺序：

```
memory {
  memory.limit_in_bytes = 2G;
  memory.memsw.limit_in_bytes = 4G;
}
```

memory.oom_control

可以为 cgroup 启用或者禁用“内存不足”（Out of Memory，OOM）终止程序。

0表示启用，尝试消耗超过其允许内存的任务会被 OOM 终止程序立即终止。默认所有使用 memory 子系统的 cgroup 都会启用 OOM 终止程序。如果要禁用，则设置 `memory.oom_control` 为 1。

memory.swappiness

设置使用swap的倾向优先级。与 `/proc/sys/vm/swappiness` 作用类似，只是这里设置的是cgroup范围的。默认值为 60。低于 60 会降低内核换出进程内存的倾向；高于 0 会增加内核换出进程内存的倾向。高于 100 时，kernel 将开始换出作为该 cgroup 中进程地址空间一部分的页面。

注意：值 0 不会阻止进程内存被换出；系统内存不足时，换出仍可能发生，因为全局虚拟内存管理逻辑不读取该 cgroup 值。

统计报告类：

memory.usage_in_bytes

报告 cgroup 中进程当前所用的内存总量（以字节为单位）。

memory.memsw.usage_in_bytes

报告该 cgroup 中进程当前所用的内存量和 swap 空间总和（以字节为单位）。

memory.max_usage_in_bytes

报告 cgroup 中进程所用的最大内存量（以字节为单位）。

memory.memsw.max_usage_in_bytes

报告该 cgroup 中进程的最大内存用量和最大 swap 空间用量（以字节为单位）。

六、参考

1. 资源管理指南
2. Resource Management Guide