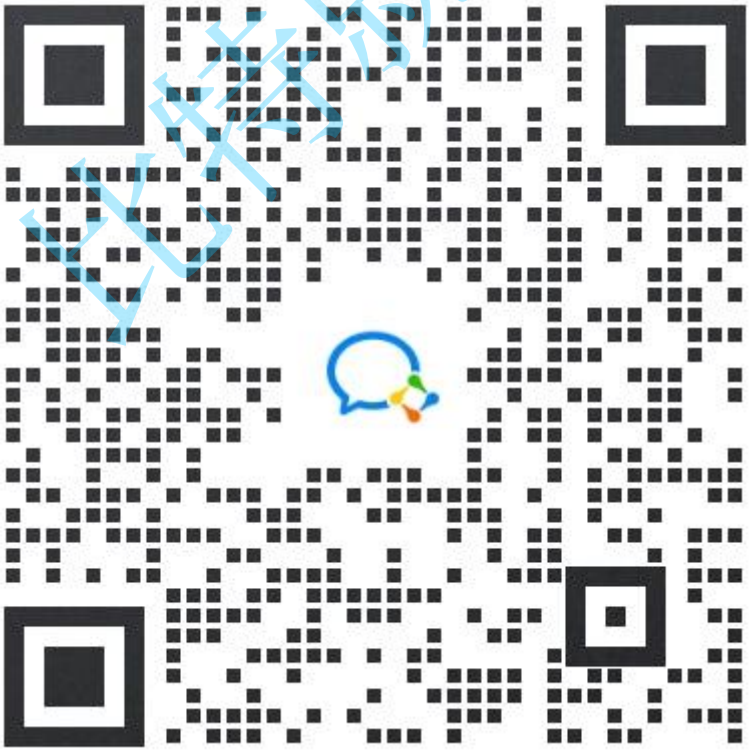
资源控制实战

版权说明

对比特项目感兴趣，可以联系这个微信。



CGroups 资源控制实战

实战目的

能够查看 cgroups 基本信息，了解容器化中，操作系统是真正的资源控制层，对 cpu 和内存的控制有一定的理解。

基础知识

pidstat

- 概述

pidstat 是 sysstat 的一个命令，用于监控全部或指定进程的 CPU、内存、线程、设备 IO 等系统资源的占用情况。Pidstat 第一次采样显示自系统启动开始的各项统计信息，后续采样将显示自上次运行命令后的统计信息。用户可以通过指定统计的次数和时间来获得所需的统计信息。

- 语法

```
Shell
pidstat [ 选项 ] [ <时间间隔> ] [ <次数> ]
```

- 参数

-u：默认参数，显示各进程的 CPU 使用统计

-r：显示各进程的内存使用统计

-d：显示各进程的 IO 使用情况

-p：指定进程号,ALL 表示所有进程

-C：指定命令

-l：显示命令名和所有参数

- 安装

Ubuntu 安装

```
Shell
#卸载
apt remove sysstat -y
#安装
apt install sysstat -y
```

CentOS 安装

```Shell
#卸载
yum remove sysstat -y
#安装
yum install sysstat -y
```

stress

- 概述

stress 是 Linux 的一个压力测试工具，可以对 CPU、Memory、IO、磁盘进行压力测试。

- 语法

```Shell
stress [OPTION [ARG]]
```

- 参数

-c, --cpu N：产生 N 个进程，每个进程都循环调用 sqrt 函数产生 CPU 压力。
-i, --io N：产生 N 个进程，每个进程循环调用 sync 将内存缓冲区内容写到磁盘上，产生 IO 压力。通过系统调用 sync 刷新内存缓冲区数据到磁盘中，以确保同步。如果缓冲区内数据较少，写到磁盘中的数据也较少，不会产生 IO 压力。在 SSD 磁盘环境中尤为明显，很可能 iowait 总是 0，却因为大量调用系统调用 sync，导致系统 CPU 使用率 sys 升高。
-m, --vm N：产生 N 个进程，每个进程循环调用 malloc/free 函数分配和释放内存。
　--vm-bytes B：指定分配内存的大小
　--vm-keep：一直占用内存，区别于不断的释放和重新分配(默认是不断释放并重新分配内存)
-d, --hdd N：产生 N 个不断执行 write 和 unlink 函数的进程(创建文件，写入内容，删除文件)
　--hdd-bytes B：指定文件大小
-t, --timeout N：在 N 秒后结束程序
-q, --quiet：程序在运行的过程中不输出信息

- 安装

Ubuntu:

```Shell
#卸载
apt remove stress -y
```

```
#安装
apt install stress -y
```

CentOS:

```
Shell
#卸载
yum remove stress -y
#安装
yum install stress -y
```

实操一、cgroups 信息查看

cgroups 版本查看

```
Shell
root@139-159-150-152:/sys/fs/cgroup# cat /proc/filesystems |grep
cgroup
nodev        cgroup
nodev        cgroup2
```

如果看到 cgroup2，表示支持 cgroup v2

cgroups 子系统查看

```
Shell
root@139-159-150-152:/sys/fs/cgroup# cat /proc/cgroups
#subsys_name        hierarchy        num_cgroups        enabled
cpuset        8        2        1
cpu        2        100        1
cpuacct        2        100        1
blkio        6        98        1
memory        10        124        1
devices        12        98        1
freezer        7        2        1
net_cls        4        2        1
perf_event        3        2        1
net_prio        4        2        1
hugetlb        9        2        1
pids        5        107        1
```

```
rdma        11        2        1
```

cgroups 挂载信息查看

可以看到默认存储位置为 /sys/fs/cgroup

```
Shell
root@139-159-150-152:/sys/fs/cgroup# mount |grep cgroup
tmpfs on /sys/fs/cgroup type tmpfs
(ro,nosuid,nodev,noexec,mode=755)
cgroup2 on /sys/fs/cgroup/unified type cgroup2
(rw,nosuid,nodev,noexec,relatime,nsdelegate)
cgroup on /sys/fs/cgroup/systemd type cgroup
(rw,nosuid,nodev,noexec,relatime,xattr,name=systemd)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup
(rw,nosuid,nodev,noexec,relatime,cpu,cpuacct)
cgroup on /sys/fs/cgroup/perf_event type cgroup
(rw,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup
(rw,nosuid,nodev,noexec,relatime,net_cls,net_prio)
cgroup on /sys/fs/cgroup/pids type cgroup
(rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/blkio type cgroup
(rw,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/freezer type cgroup
(rw,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/cpuset type cgroup
(rw,nosuid,nodev,noexec,relatime,cpuset)
cgroup on /sys/fs/cgroup/hugetlb type cgroup
(rw,nosuid,nodev,noexec,relatime,hugetlb)
cgroup on /sys/fs/cgroup/memory type cgroup
(rw,nosuid,nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/rdma type cgroup
(rw,nosuid,nodev,noexec,relatime,rdma)
cgroup on /sys/fs/cgroup/devices type cgroup
(rw,nosuid,nodev,noexec,relatime,device
```

查看一个进程上的 cgroup 限制

1. 以当前 shell 进程为例,查看进程的 cgroup

```
Shell
[root@VM-8-12-centos ~]# cat /proc/$$/cgroup
```

```
11:hugetlb:/
10:memory:/user.slice
9:freezer:/
8:cpuset:/
7:perf_event:/
6:net_prio,net_cls:/
5:devices:/user.slice/user-0.slice
4:pids:/user.slice
3:cpuacct,cpu:/user.slice
2:blkio:/user.slice
1:name=systemd:/user.slice/user-0.slice/session-354304.scope
```

2. 比如 cpu 在 user.slice，我们可以找到这个目录，里面有对 init 进程的详细限制信息

```
Shell
[root@VM-8-12-centos ~]# ll /sys/fs/cgroup/cpu/user.slice/
total 0
-rw-r--r-- 1 root root 0 Nov 21 15:30 cgroup.clone_children
--w--w--w- 1 root root 0 Nov 21 15:30 cgroup.event_control
-rw-r--r-- 1 root root 0 Nov 21 15:30 cgroup.procs
-r--r--r-- 1 root root 0 Nov 21 15:30 cpuacct.stat
-rw-r--r-- 1 root root 0 Nov 21 15:30 cpuacct.usage
-r--r--r-- 1 root root 0 Nov 21 15:30 cpuacct.usage_percpu
-rw-r--r-- 1 root root 0 Nov 21 15:30 cpu.cfs_period_us
-rw-r--r-- 1 root root 0 Nov 21 15:30 cpu.cfs_quota_us
-rw-r--r-- 1 root root 0 Nov 21 15:30 cpu.rt_period_us
-rw-r--r-- 1 root root 0 Nov 21 15:30 cpu.rt_runtime_us
-rw-r--r-- 1 root root 0 Nov 21 15:30 cpu.shares
-r--r--r-- 1 root root 0 Nov 21 15:30 cpu.stat
-rw-r--r-- 1 root root 0 Nov 21 15:30 notify_on_release
-rw-r--r-- 1 root root 0 Nov 21 15:30 tasks
```

实操二、使用 cgroups 对内存进行控制

1. 创建内存的 cgroup，很简单我们进入到 cgroup 的内存控制目录 /sys/fs/cgroup/memory，我们创建目录 test_memory

```
Shell
root@139-159-150-152:/sys/fs/cgroup/memory# mkdir test_memory
root@139-159-150-152:/sys/fs/cgroup/memory# ll
```

```
total 0
dr-xr-xr-x  7 root root   0 Mar 10 14:13 ./
drwxr-xr-x 15 root root 380 Mar 10 14:13 ../
-rw-r--r--  1 root root   0 Mar 12 14:13 cgroup.clone_children
--w--w--w-  1 root root   0 Mar 12 14:13 cgroup.event_control
-rw-r--r--  1 root root   0 Mar 12 14:13 cgroup.procs
-r--r--r--  1 root root   0 Mar 12 14:13 cgroup.sane_behavior
drwxr-xr-x  2 root root   0 Mar 12 14:13 docker/
drwxr-xr-x  2 root root   0 Mar 12 14:13 init.scope/
-rw-r--r--  1 root root   0 Mar 12 14:13 memory.failcnt
--w-------  1 root root   0 Mar 12 14:13 memory.force_empty
-rw-r--r--  1 root root   0 Mar 12 14:13 memory.kmem.failcnt
-rw-r--r--  1 root root   0 Mar 12 14:13
memory.kmem.limit_in_bytes
-rw-r--r--  1 root root   0 Mar 12 14:13
memory.kmem.max_usage_in_bytes
-r--r--r--  1 root root   0 Mar 12 14:13 memory.kmem.slabinfo
-rw-r--r--  1 root root   0 Mar 12 14:13 memory.kmem.tcp.failcnt
-rw-r--r--  1 root root   0 Mar 12 14:13
memory.kmem.tcp.limit_in_bytes
-rw-r--r--  1 root root   0 Mar 12 14:13
memory.kmem.tcp.max_usage_in_bytes
-r--r--r--  1 root root   0 Mar 12 14:13
memory.kmem.tcp.usage_in_bytes
-r--r--r--  1 root root   0 Mar 12 14:13
memory.kmem.usage_in_bytes
-rw-r--r--  1 root root   0 Mar 12 14:13 memory.limit_in_bytes
-rw-r--r--  1 root root   0 Mar 12 14:13 memory.max_usage_in_bytes
-rw-r--r--  1 root root   0 Mar 12 14:13
memory.move_charge_at_immigrate
-r--r--r--  1 root root   0 Mar 12 14:13 memory.numa_stat
-rw-r--r--  1 root root   0 Mar 12 14:13 memory.oom_control
----------  1 root root   0 Mar 12 14:13 memory.pressure_level
-rw-r--r--  1 root root   0 Mar 12 14:13
memory.soft_limit_in_bytes
-r--r--r--  1 root root   0 Mar 12 14:13 memory.stat
-rw-r--r--  1 root root   0 Mar 12 14:13 memory.swappiness
-r--r--r--  1 root root   0 Mar 12 14:13 memory.usage_in_bytes
-rw-r--r--  1 root root   0 Mar 12 14:13 memory.use_hierarchy
-rw-r--r--  1 root root   0 Mar 12 14:13 notify_on_release
-rw-r--r--  1 root root   0 Mar 12 14:13 release_agent
drwxr-xr-x 95 root root   0 Mar 12 13:22 system.slice/
-rw-r--r--  1 root root   0 Mar 12 14:13 tasks
drwxr-xr-x  2 root root   0 Mar 12 14:14 test_memory/
```

```
drwxr-xr-x  3 root root   0 Mar 12 13:58 user.slice/
```

2. 可以看到内存限制文件已经自动在 test_memory 中创建完成了，cgroups 文件系统会在创建文件目录的时候自动创建相应的配置文件

```
Shell
root@139-159-150-152:/sys/fs/cgroup/memory/test_memory# ll
total 0
drwxr-xr-x 2 root root 0 Mar 12 14:14 ./
dr-xr-xr-x 7 root root 0 Mar 10 14:13 ../
-rw-r--r-- 1 root root 0 Mar 12 14:14 cgroup.clone_children
--w--w--w- 1 root root 0 Mar 12 14:14 cgroup.event_control
-rw-r--r-- 1 root root 0 Mar 12 14:14 cgroup.procs
-rw-r--r-- 1 root root 0 Mar 12 14:14 memory.failcnt
--w------- 1 root root 0 Mar 12 14:14 memory.force_empty
-rw-r--r-- 1 root root 0 Mar 12 14:14 memory.kmem.failcnt
-rw-r--r-- 1 root root 0 Mar 12 14:14 memory.kmem.limit_in_bytes
-rw-r--r-- 1 root root 0 Mar 12 14:14
memory.kmem.max_usage_in_bytes
-r--r--r-- 1 root root 0 Mar 12 14:14 memory.kmem.slabinfo
-rw-r--r-- 1 root root 0 Mar 12 14:14 memory.kmem.tcp.failcnt
-rw-r--r-- 1 root root 0 Mar 12 14:14
memory.kmem.tcp.limit_in_bytes
-rw-r--r-- 1 root root 0 Mar 12 14:14
memory.kmem.tcp.max_usage_in_bytes
-r--r--r-- 1 root root 0 Mar 12 14:14
memory.kmem.tcp.usage_in_bytes
-r--r--r-- 1 root root 0 Mar 12 14:14 memory.kmem.usage_in_bytes
-rw-r--r-- 1 root root 0 Mar 12 14:14 memory.limit_in_bytes
-rw-r--r-- 1 root root 0 Mar 12 14:14 memory.max_usage_in_bytes
-rw-r--r-- 1 root root 0 Mar 12 14:14
memory.move_charge_at_immigrate
-r--r--r-- 1 root root 0 Mar 12 14:14 memory.numa_stat
-rw-r--r-- 1 root root 0 Mar 12 14:14 memory.oom_control
---------- 1 root root 0 Mar 12 14:14 memory.pressure_level
-rw-r--r-- 1 root root 0 Mar 12 14:14 memory.soft_limit_in_bytes
-r--r--r-- 1 root root 0 Mar 12 14:14 memory.stat
-rw-r--r-- 1 root root 0 Mar 12 14:14 memory.swappiness
-r--r--r-- 1 root root 0 Mar 12 14:14 memory.usage_in_bytes
-rw-r--r-- 1 root root 0 Mar 12 14:14 memory.use_hierarchy
-rw-r--r-- 1 root root 0 Mar 12 14:14 notify_on_release
-rw-r--r-- 1 root root 0 Mar 12 14:14 tasks
```

3. 配置 cgroup 的策略为最大使用 20M 内存

```
Shell
root@139-159-150-152:/sys/fs/cgroup/memory# expr 20 \* 1024 \*
1024
20971520
root@139-159-150-152:/sys/fs/cgroup/memory# cat
test_memory/memory.limit_in_bytes
9223372036854771712
root@139-159-150-152:/sys/fs/cgroup/memory# echo "20971520" >
test_memory/memory.limit_in_bytes
root@139-159-150-152:/sys/fs/cgroup/memory# cat
test_memory/memory.limit_in_bytes
20971520
```

4. 启动 1 个消耗内存的进程，每个进程占用 50M 内存

```
Shell
root@139-159-150-152:/sys/fs/cgroup# stress -m 1 --vm-bytes 50M
stress: info: [62106] dispatching hogs: 0 cpu, 0 io, 1 vm, 0 hdd
```

5. 打开一个新的 shell 窗口 B 窗口，使用 pidstat 查看状态，红色为进程 id

```
Shell
root@139-159-150-152:/sys/fs/cgroup/memory# pidstat -r -C stress -
p ALL 1 10000
Linux 5.4.0-100-generic (139-159-150-152)         03/12/2023
_x86_64_        (1 CPU)

02:47:01 PM   UID       PID  minflt/s  majflt/s     VSZ
RSS   %MEM  Command
02:47:02 PM     0     62517      0.00      0.00    3856     988
0.05  stress
02:47:02 PM     0     62518 476597.03      0.00   55060   15156
0.75  stress

02:47:02 PM   UID       PID  minflt/s  majflt/s     VSZ  echo
62518 > test_memory/tasks    RSS   %MEM  Command
02:47:03 PM     0     62517      0.00      0.00    3856     988
0.05  stress
02:47:03 PM     0     62518 483459.00      0.00   55060    3540
0.17  stress
# 注意红色为 PID
02:47:03 PM   UID       PID  minflt/s  majflt/s     VSZ
RSS   %MEM  Command
```

```
02:47:04 PM     0     62517      0.00      0.00    3856     988
0.05  stress
02:47:04 PM     0     62518 489336.00      0.00   55060   15156
0.75  stress
```

6. 打开一个新的 shell C 窗口，将进程 id 移动到我们的 cgroup 策略

```
Shell
cd /sys/fs/cgroup/memory
echo 62518 >> test_memory/tasks
```

7. 可以看到进程无法申请到足够内存退出

```
root@139-159-150-152:/sys/fs/cgroup# stress -m 1 --vm-bytes 50M
stress: info: [62517] dispatching hogs: 0 cpu, 0 io, 1 vm, 0 hdd
stress: FAIL: [62517] (415) <-- worker 62518 got signal 9
stress: WARN: [62517] (417) now reaping child worker processes
stress: FAIL: [62517] (451) failed run completed in 176s
root@139-159-150-152:/sys/fs/cgroup#
```

可以看到进程消失了

```
02:49:16 PM   UID       PID  minflt/s  majflt/s     VSZ     RSS   %MEM  Command
02:49:17 PM     0     62517      0.00      0.00    3856     988   0.05  stress
02:49:17 PM     0     62518 483216.00      0.00   55060   26244   1.29  stress

02:49:17 PM   UID       PID  minflt/s  majflt/s     VSZ     RSS   %MEM  Command

02:49:18 PM   UID       PID  minflt/s  majflt/s     VSZ     RSS   %MEM  Command

02:49:19 PM   UID       PID  minflt/s  majflt/s     VSZ     RSS   %MEM  Command
```

实操三、使用 cgroups 对 cpu 进行控制

1. 创建内存的 cgroup，很简单我们进入到 cgroup 的内存控制目录 /sys/fs/cgroup/cpu，我们创建目录 test_cpu，可以看到系统会自动为我们创建 cgroup 的 cpu 策略

```
Shell
root@139-159-150-152:/sys/fs/cgroup/cpu# cd /sys/fs/cgroup/cpu
root@139-159-150-152:/sys/fs/cgroup/cpu# mkdir test_cpu
root@139-159-150-152:/sys/fs/cgroup/cpu# ll test_cpu
total 0
drwxr-xr-x 2 root root 0 Mar 12 14:54 ./
dr-xr-xr-x 9 root root 0 Mar 10 14:13 ../
-rw-r--r-- 1 root root 0 Mar 12 14:54 cgroup.clone_children
-rw-r--r-- 1 root root 0 Mar 12 14:54 cgroup.procs
-r--r--r-- 1 root root 0 Mar 12 14:54 cpuacct.stat
```

```
-rw-r--r-- 1 root root 0 Mar 12 14:54 cpuacct.usage
-r--r--r-- 1 root root 0 Mar 12 14:54 cpuacct.usage_all
-r--r--r-- 1 root root 0 Mar 12 14:54 cpuacct.usage_percpu
-r--r--r-- 1 root root 0 Mar 12 14:54 cpuacct.usage_percpu_sys
-r--r--r-- 1 root root 0 Mar 12 14:54 cpuacct.usage_percpu_user
-r--r--r-- 1 root root 0 Mar 12 14:54 cpuacct.usage_sys
-r--r--r-- 1 root root 0 Mar 12 14:54 cpuacct.usage_user
-rw-r--r-- 1 root root 0 Mar 12 14:54 cpu.cfs_period_us
-rw-r--r-- 1 root root 0 Mar 12 14:54 cpu.cfs_quota_us
-rw-r--r-- 1 root root 0 Mar 12 14:54 cpu.shares
-r--r--r-- 1 root root 0 Mar 12 14:54 cpu.stat
-rw-r--r-- 1 root root 0 Mar 12 14:54 cpu.uclamp.max
-rw-r--r-- 1 root root 0 Mar 12 14:54 cpu.uclamp.min
-rw-r--r-- 1 root root 0 Mar 12 14:54 notify_on_release
-rw-r--r-- 1 root root 0 Mar 12 14:54 tasks
```

2. 打开新的 shell 窗口 B 窗口，使用 stress 模拟一个任务，cpu 使用率为 100

```
Shell
root@139-159-150-152:/sys/fs/cgroup# stress -c 1
stress: info: [62576] dispatching hogs: 1 cpu, 0 io, 0 vm, 0 hdd
```

3. 可以看到 cpu 的使用率为 100%

```
Shell
root@139-159-150-152:/sys/fs/cgroup/memory# pidstat -u -C stress -
p ALL 1 10000
Linux 5.4.0-100-generic (139-159-150-152)         03/12/2023
_x86_64_        (1 CPU)

02:59:38 PM   UID
PID    %usr %system  %guest   %wait    %CPU   CPU  Command
02:59:39 PM     0     62576    0.00    0.00    0.00    0.00
0.00     0  stress
02:59:39 PM     0     62577   99.01    0.00    0.00    0.99
99.01     0  stress

02:59:39 PM   UID
PID    %usr %system  %guest   %wait    %CPU   CPU  Command
02:59:40 PM     0     62576    0.00    0.00    0.00    0.00
0.00     0  stress
02:59:40 PM     0     62577   99.00    0.00    0.00    1.00
```

```
99.00      0  stress
```

4. 打开新的 shell 窗口 C 窗口，我们设置 cproup 的 cpu 使用率为 30%，cpu 使用率的计算公式 **cfs_quota_us/cfs_period_us**

1）**cfs_period_us**：cfs_period_us 表示一个 cpu 带宽，单位为微秒。系统总 CPU 带宽，默认值 100000。

2）**cfs_quota_us**：cfs_quota_us 表示 Cgroup 可以使用的 cpu 的带宽，单位为微秒。cfs_quota_us 为-1，表示使用的 CPU 不受 cgroup 限制。cfs_quota_us 的最小值为 1ms(1000)，最大值为 1s。

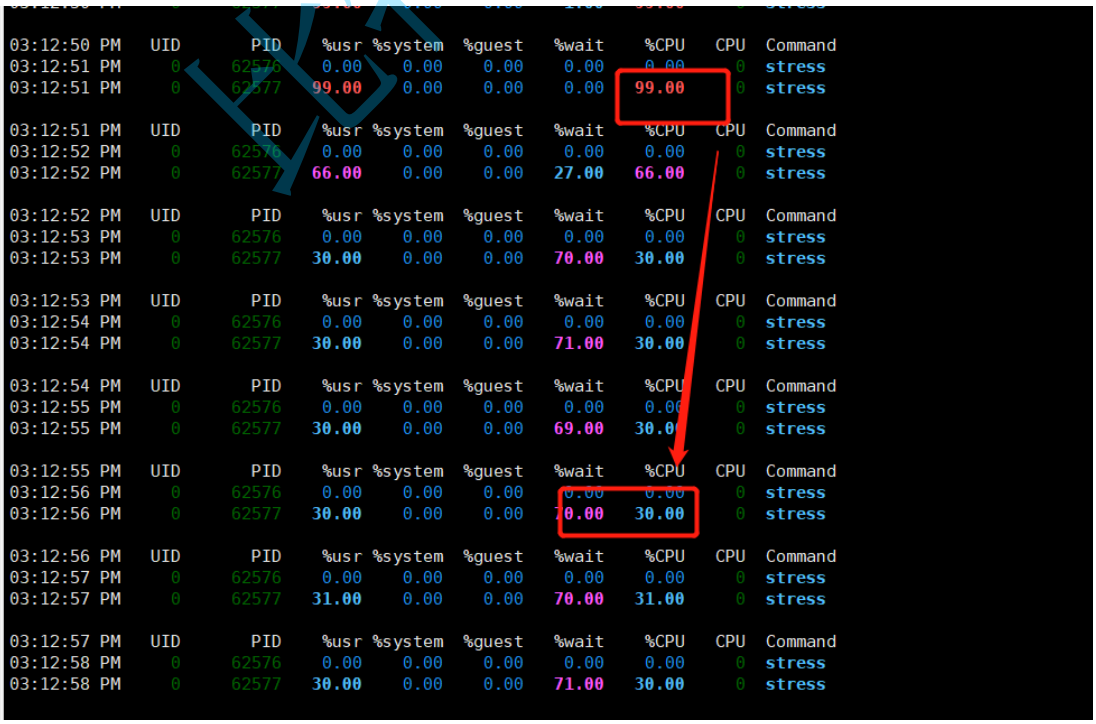
所以我们将 **cfs_quota_us 的值设置为 `30000`，从理论上讲就可以限制 test_cpu 控制的进程的 cpu 利用率最多是 30%。**

```
Shell
 cd /sys/fs/cgroup/cpu
echo 30000 > test_cpu/cpu.cfs_quota_us
```

5. 我们可以看到进程的 PID 为 62577，我们将该进程放到 tasks 文件进行控制

```
Shell
cd /sys/fs/cgroup/cpu
echo 62577 > test_cpu/tasks
```

6. B 窗口中可以看到我们监控的 cpu 的使用率由 100%降低为 30%



至此我们成功的模拟了对内存和 cpu 的使用控制，而 docker 本质也是调用这些的 API 来完成对资源的管理，只不过 docker 的易用性和镜像的设计更加人性化，所以 docker 才能风靡全球，docker 课程完后我们会看下 docker 如何对资源控制对比这种控制可以说简单不止一倍。