# JStorm 资源隔离测试

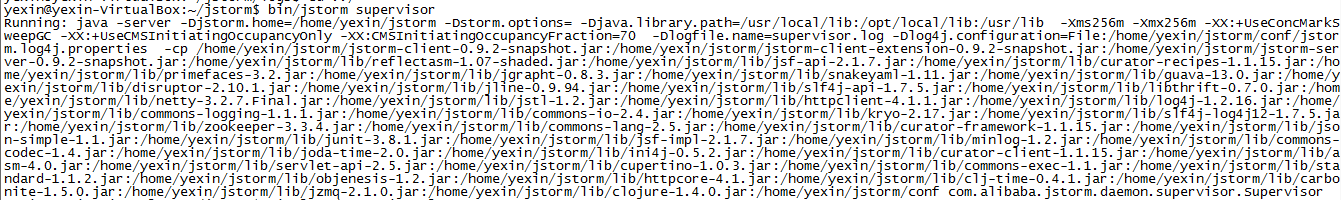
（以下测试中, supervisor.enable.cgroup: true）

## 非root权限启动supervisor

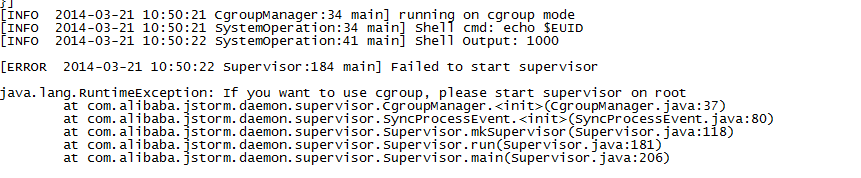
### 测试目标

非root情况,supervisor无法启动

### 测试结果



Supervisor无法启动,提示需要在root下启动

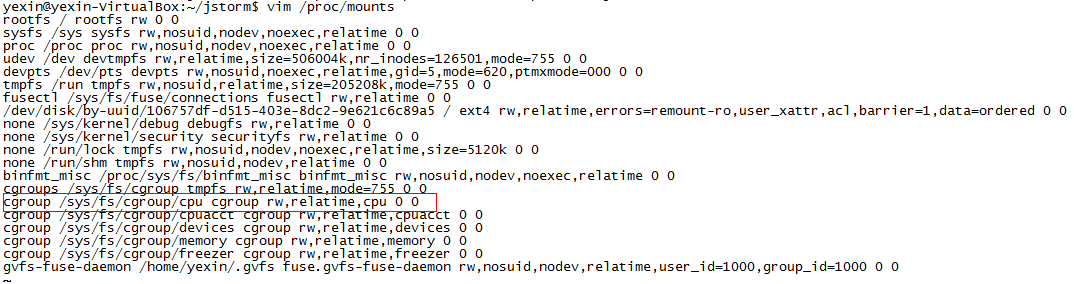


## 在已挂载CPU子系统的情况下启动supervisor

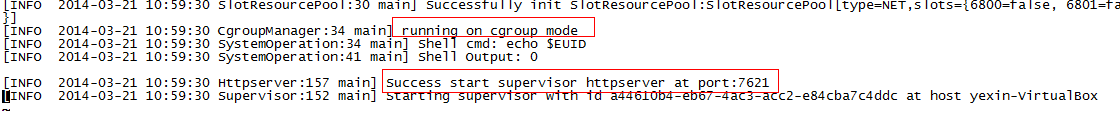
### 测试目标

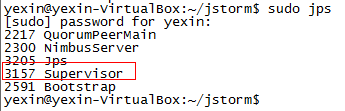
正确读取到已挂载的层级,初始化,并在层级下建立rootCgroup

### 测试结果

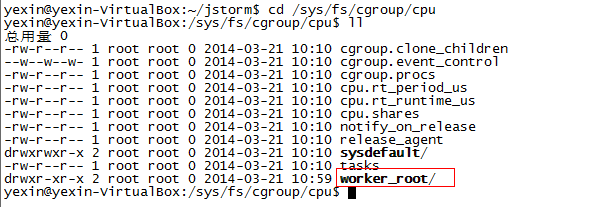


运行后:





Supervisor启动成功



rootCgroup建立成功

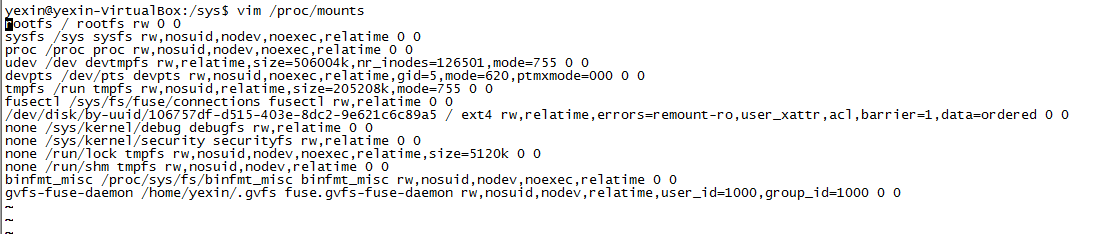
## 在未挂载CPU子系统的情况下启动supervisor

### 测试目标

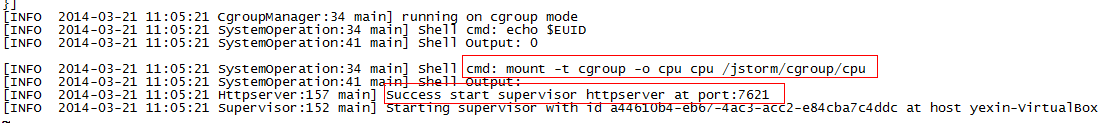
自动挂载CPU子系统到自定义层级上,并建立rootCgroup

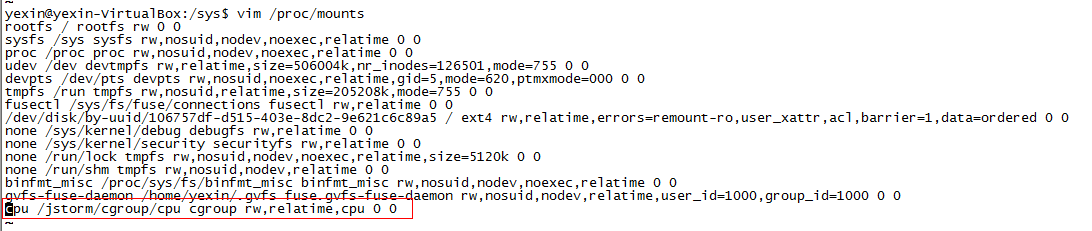
### 测试结果

未挂载cpu

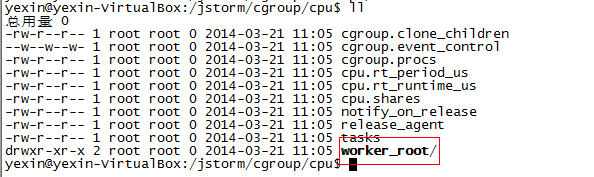


启动supervisor:





挂载层级成功



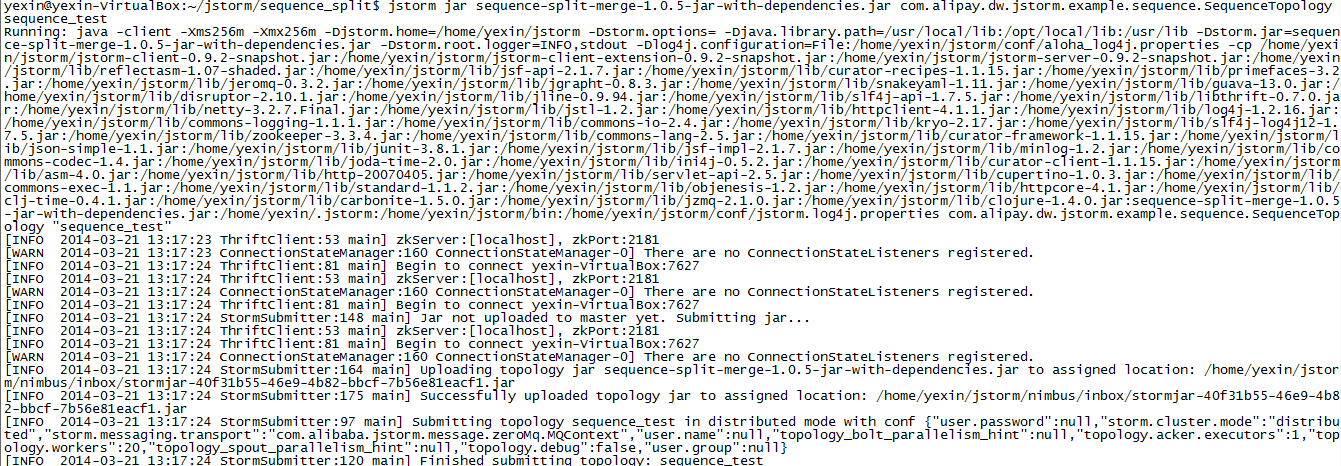
rootCgroup建立

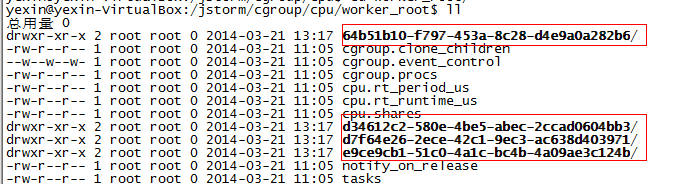
## 提交一个topology

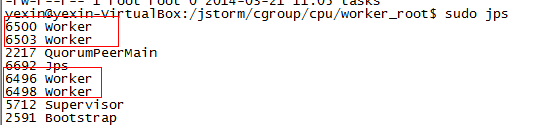
### 测试目标

Worker是否正确地被加入正确的cgroup中

### 测试结果









Worker成功地被添加到cgroup中,运行正常



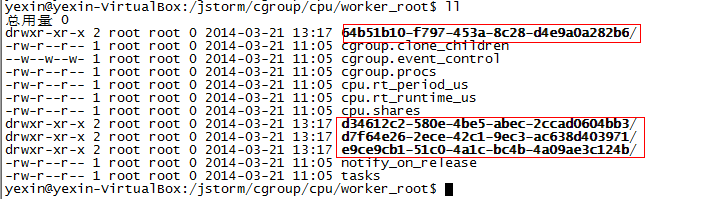
占用一个cpu slot,cpu weight为1024

## 杀死一个topology

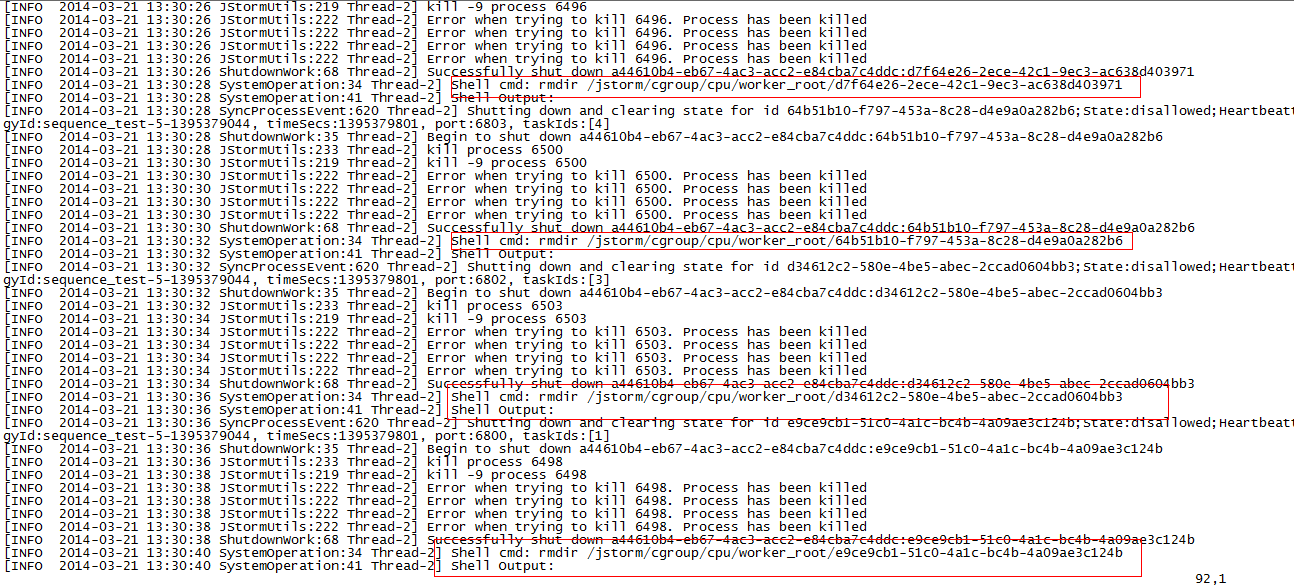
### 测试目标

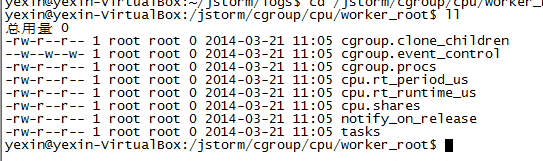
Cgroup是否被正确删除

### 测试结果









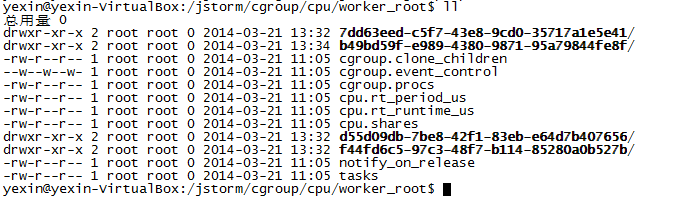
Workers的Cgroup被正确删除

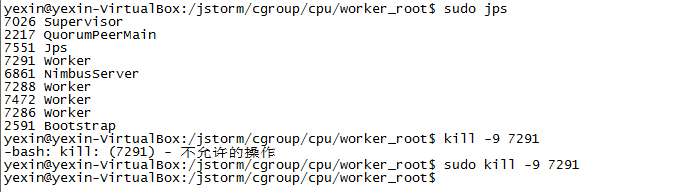
## 杀死一个worker

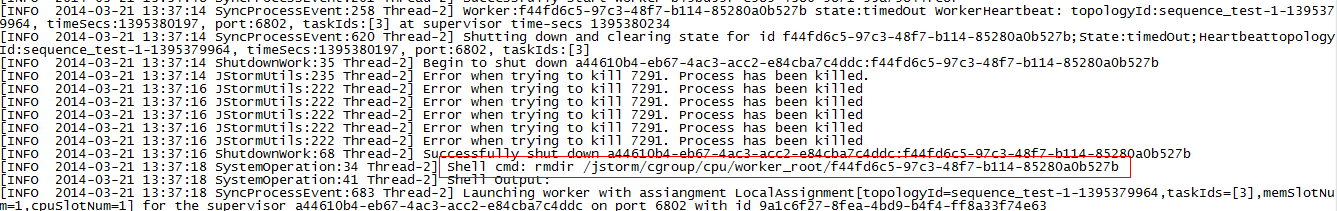
### 测试目标

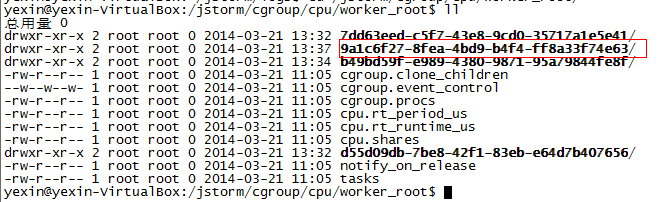
Supervisor能否正确回收被杀死worker的cgroup,并且重新启动worker将worker加入新的cgroup重

### 测试结果









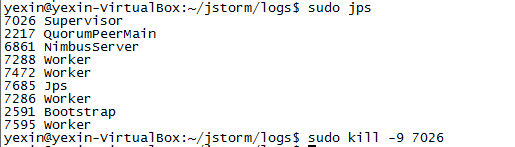
死去worker的cgroup被成功删除,新的worker的cgroup被成功创建,worker加入其中

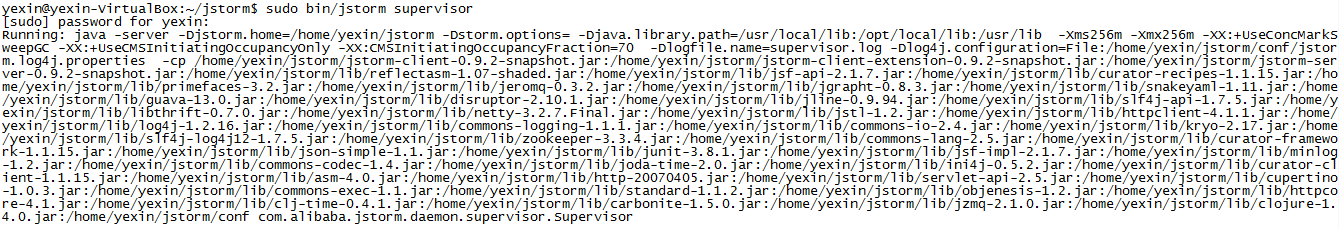
## 提交topology，杀死supervisor，重启supervisor后杀死topology

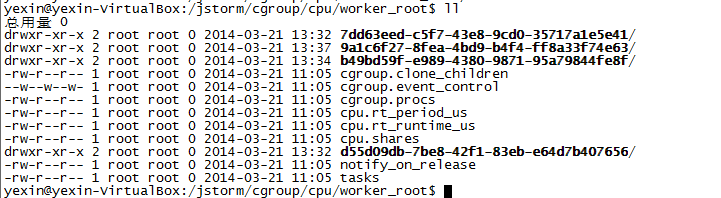
### 测试目标

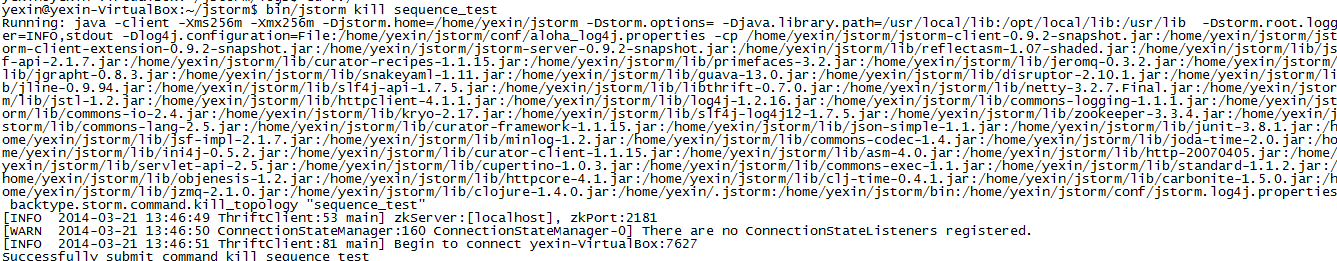
Supervisor能否正确读取当前状态的cgroup,并且正确的清除当前worker的cgroup

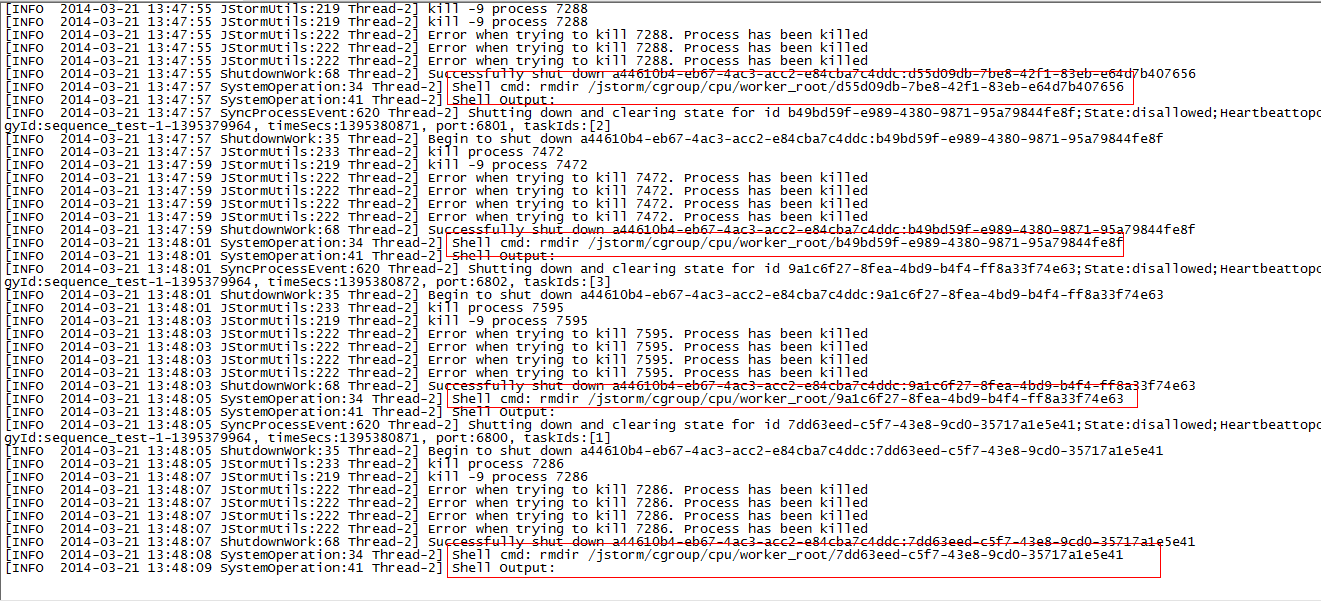
### 测试结果

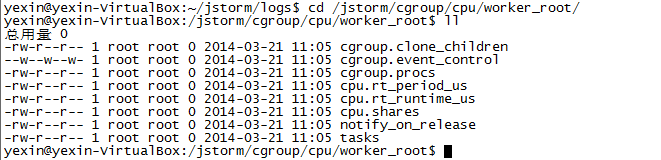












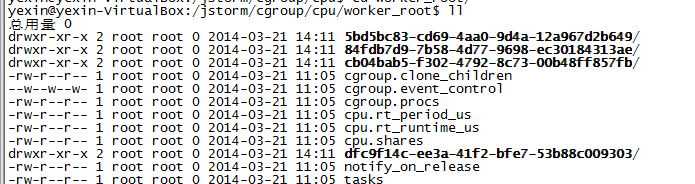
Worker的cgroup被正确地清除了

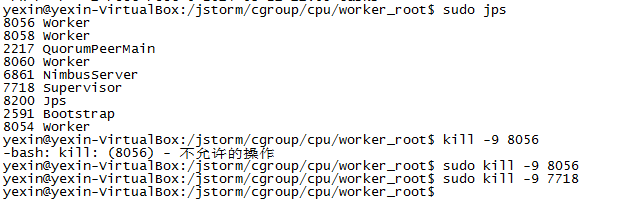
## 杀死正在运行的worker,然后马上杀死supervisor再重启

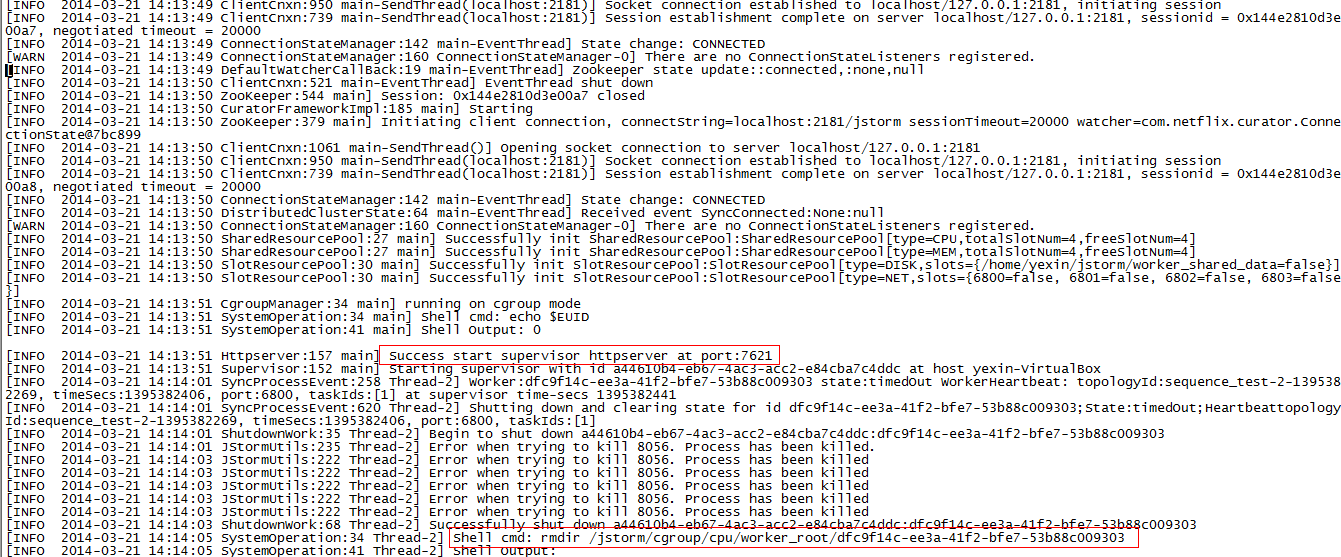
### 测试目标

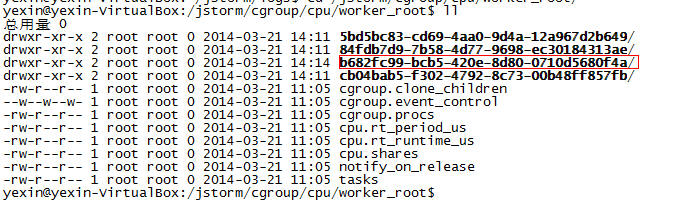
重启后的supervisor是否能正确清理cgroup，并且重启worker,放入新的cgroup中

### 测试结果









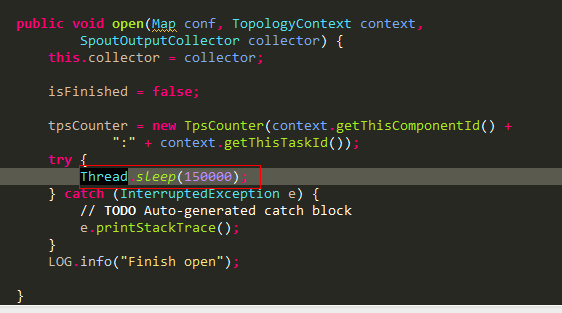
测试成功,旧的cgroup被清理,新的cgroup被创建,新的worker被创建加入其中

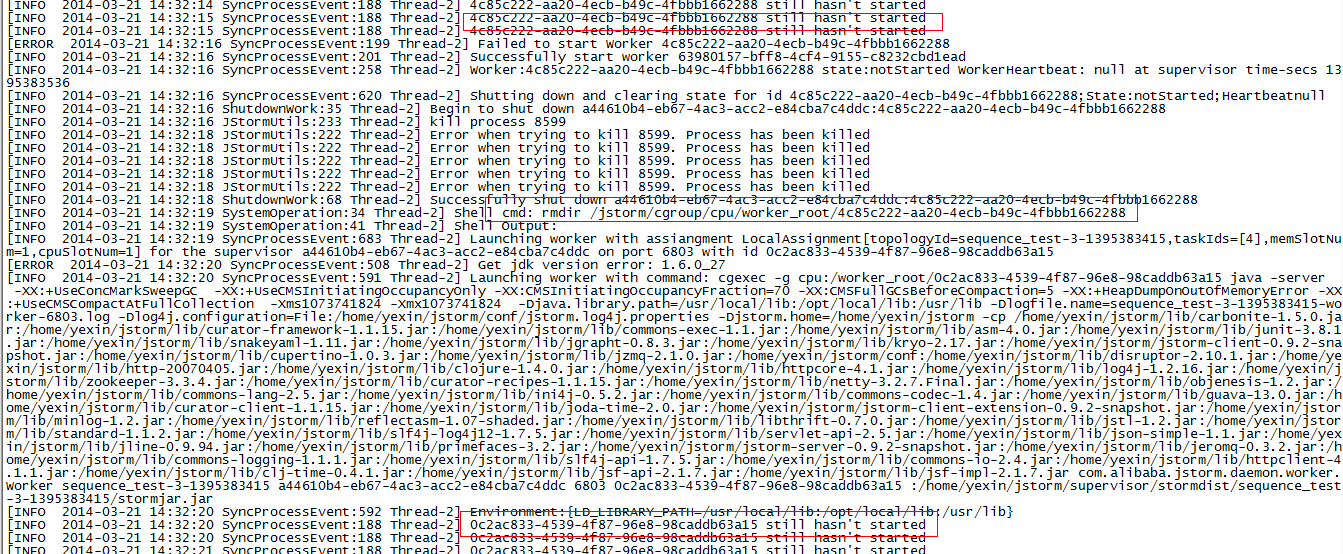
## 在spout open过程中sleep超过120s

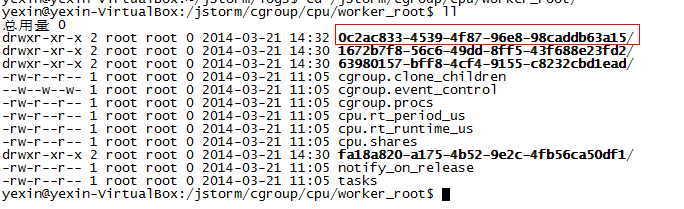
### 测试目标

在worker因为错误没有被正确启动,supervisor会不断尝试重启worker,测试在这种情况下cgroup是否被正确清除.(这里选择的是在open spout时sleep超过120s,supervisor会认为spout启动失败)

### 测试结果







超时后旧的cgroup被正确清除，新的cgroup被正确创建