

# Process Number Controller

## Abstract

The process number controller is used to allow a cgroup hierarchy to stop any new tasks from being fork()'d or clone()'d after a certain limit is reached.

Since it is trivial to hit the task limit without hitting any kmemcg limits in place, PIDs are a fundamental resource. As such, PID exhaustion must be preventable in the scope of a cgroup hierarchy by allowing resource limiting of the number of tasks in a cgroup.

## Usage

In order to use the *pids* controller, set the maximum number of tasks in *pids.max* (this is not available in the root cgroup for obvious reasons). The number of processes currently in the cgroup is given by *pids.current*.

Organisational operations are not blocked by cgroup policies, so it is possible to have *pids.current* > *pids.max*. This can be done by either setting the limit to be smaller than *pids.current*, or attaching enough processes to the cgroup such that *pids.current* > *pids.max*. However, it is not possible to violate a cgroup policy through *fork()* or *clone()*. *fork()* and *clone()* will return -EAGAIN if the creation of a new process would cause a cgroup policy to be violated.

To set a cgroup to have no limit, set *pids.max* to "max". This is the default for all new cgroups (N.B. that PID limits are hierarchical, so the most stringent limit in the hierarchy is followed).

*pids.current* tracks all child cgroup hierarchies, so *parent/pids.current* is a superset of *parent/child/pids.current*.

The *pids.events* file contains event counters:

- *max*: Number of times fork failed because limit was hit.

## Example

First, we mount the *pids* controller:

```
# mkdir -p /sys/fs/cgroup/pids
# mount -t cgroup -o pids none /sys/fs/cgroup/pids
```

Then we create a hierarchy, set limits and attach processes to it:

```
# mkdir -p /sys/fs/cgroup/pids/parent/child
# echo 2 > /sys/fs/cgroup/pids/parent/pids.max
# echo $$ > /sys/fs/cgroup/pids/parent/cgroup.procs
# cat /sys/fs/cgroup/pids/parent/pids.current
2
#
```

It should be noted that attempts to overcome the set limit (2 in this case) will fail:


```
# cat /sys/fs/cgroup/pids/parent/pids.current
2
# ( /bin/echo "Here's some processes for you." | cat )
sh: fork: Resource temporary unavailable
#
```

Even if we migrate to a child cgroup (which doesn't have a set limit), we will not be able to overcome the most stringent limit in the hierarchy (in this case, parent's):

```
# echo $$ > /sys/fs/cgroup/pids/parent/child/cgroup.procs
# cat /sys/fs/cgroup/pids/parent/pids.current
2
# cat /sys/fs/cgroup/pids/parent/child/pids.current
2
# cat /sys/fs/cgroup/pids/parent/child/pids.max
max
# ( /bin/echo "Here's some processes for you." | cat )
sh: fork: Resource temporary unavailable
#
```

We can set a limit that is smaller than *pids.current*, which will stop any new processes from being forked at all (note that the shell itself counts towards *pids.current*):

```
# echo 1 > /sys/fs/cgroup/pids/parent/pids.max
# /bin/echo "We can't even spawn a single process now."
sh: fork: Resource temporary unavailable
# echo 0 > /sys/fs/cgroup/pids/parent/pids.max
```



```
# /bin/echo "We can't even spawn a single process now."  
sh: fork: Resource temporary unavailable  
#
```

