Cgroup Freezer

The cgroup freezer is useful to batch job management system which start and stop sets of tasks in order to schedule the resources of a machine according to the desires of a system administrator. This sort of program is often used on HPC clusters to schedule access to the cluster as a whole. The cgroup freezer uses cgroups to describe the set of tasks to be started/stopped by the batch job management system. It also provides a means to start and stop the tasks composing the job.

The cgroup freezer will also be useful for checkpointing running groups of tasks. The freezer allows the checkpoint code to obtain a consistent image of the tasks by attempting to force the tasks in a cgroup into a quiescent state. Once the tasks are quiescent another task can walk /proc or invoke a kernel interface to gather information about the quiesced tasks. Checkpointed tasks can be restarted later should a recoverable error occur. This also allows the checkpointed tasks to be migrated between nodes in a cluster by copying the gathered information to another node and restarting the tasks there.

Sequences of SIGSTOP and SIGCONT are not always sufficient for stopping and resuming tasks in userspace. Both of these signals are observable from within the tasks we wish to freeze. While SIGSTOP cannot be caught, blocked, or ignored it can be seen by waiting or ptracing parent tasks. SIGCONT is especially unsuitable since it can be caught by the task. Any programs designed to watch for SIGSTOP and SIGCONT could be broken by attempting to use SIGSTOP and SIGCONT to stop and resume tasks. We can demonstrate this problem using nested bash shells:

```
$ echo $$
16644
$ bash
$ echo $$
16690

From a second, unrelated bash shell:
$ kill -SIGSTOP 16690
$ kill -SIGCONT 16690

<at this point 16690 exits and causes 16644 to exit too>
```

This happens because bash can observe both signals and choose how it responds to them.

Another example of a program which catches and responds to these signals is gdb. In fact any program designed to use ptrace is likely to have a problem with this method of stopping and resuming tasks.

In contrast, the cgroup freezer uses the kernel freezer code to prevent the freeze/unfreeze cycle from becoming visible to the tasks being frozen. This allows the bash example above and gdb to run as expected.

The cgroup freezer is hierarchical. Freezing a cgroup freezes all tasks belonging to the cgroup and all its descendant cgroups. Each cgroup has its own state (self-state) and the state inherited from the parent (parent-state). Iff both states are THAWED, the cgroup is THAWED.

The following cgroupfs files are created by cgroup freezer.

- freezer.state: Read-write.

  When read, returns the effective state of the cgroup - "THAWED", "FREEZING" or "FROZEN". This is the combined self and parent-states. If any is freezing, the cgroup is freezing (FREEZING or FROZEN).

  FREEZING cgroup transitions into FROZEN state when all tasks belonging to the cgroup and its descendants become frozen. Note that a cgroup reverts to FREEZING from FROZEN after a new task is added to the cgroup or one of its descendant cgroups until the new task is frozen.

  When written, sets the self-state of the cgroup. Two values are allowed - "FROZEN" and "THAWED". If FROZEN is written, the cgroup, if not already freezing, enters FREEZING state along with all its descendant cgroups.

  If THAWED is written, the self-state of the cgroup is changed to THAWED. Note that the effective state may not change to THAWED if the parent-state is still freezing. If a cgroup's effective state becomes THAWED, all its descendants which are freezing because of the cgroup also leave the freezing state.

- freezer.self_freezing: Read only.

  Shows the self-state. 0 if the self-state is THAWED; otherwise, 1. This value is 1 iff the last write to freezer.state was "FROZEN".

- freezer.parent_freezing: Read only.

  Shows the parent-state. 0 if none of the cgroup's ancestors is frozen; otherwise, 1.

The root cgroup is non-freezable and the above interface files don't exist.

- Examples of usage:

  ```
  # mkdir /sys/fs/cgroup/freezer
  # mount -t cgroup -ofreezer freezer /sys/fs/cgroup/freezer
  # mkdir /sys/fs/cgroup/freezer/0
  ```

```
# echo $some_pid > /sys/fs/cgroup/freezer/0/tasks
```

to get status of the freezer subsystem:

```
# cat /sys/fs/cgroup/freezer/0/freezer.state
THAWED
```

to freeze all tasks in the container:

```
# echo FROZEN > /sys/fs/cgroup/freezer/0/freezer.state
# cat /sys/fs/cgroup/freezer/0/freezer.state
FREEZING
# cat /sys/fs/cgroup/freezer/0/freezer.state
FROZEN
```

to unfreeze all tasks in the container:

```
# echo THAWED > /sys/fs/cgroup/freezer/0/freezer.state
# cat /sys/fs/cgroup/freezer/0/freezer.state
THAWED
```

This is the basic mechanism which should do the right thing for user space task in a simple scenario.