Source Report

@ CS353 Linux Kernel

Shen Jiyuan

5130309194

1 Basic Background

Source Code from Linux Kernel Version 4.6

2 Definition Understanding

1 in Linux

@cgroup: abbreviated from control group. It is a Linux feature that liits, accounts for, and isolates the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes.

@subsys: abbreviated from subsystem. It is a module that can be added or deleted in cgroup. It provides many action controls for cgroup and it is enccapsulated in cgroup.

2 two important structures for cgroup

@struct cgroup_subsys_state: an abstrct class used by cgroup to manage subsys.

@struct cgrp_css_link: used for describing the relation between css_set and cgroup. Because one process can be attached to different cgroup, and at the same time, one cgroup can have different processes, that is the relation is many-to-many.

3 in source code

@cgroup_destroy_locked: the first stage of cgroup destruction

@cgrp: cgroup to be destroyed.

3 Implementation Understanding

1 overview

css's make use of percpu refcnts whose killing latency shouldn't be exposed to userland and are RCU protected. Also, cgroup core needs to guarantee that css_tryget_online() won't succeed by the time ->css_offline() is invoked. To satisfy all the requirements, destruction is implemented in the following two steps.

* Step1. Verify @cgrp can be destroyed and mark it dying. Remove all userland visible parts and start killing the percpu refcnts of css's. Set up so that the next stage will be kicked off once all the percpu refcnts are confirmed to be killed.
* Step2. Invoke ->css_offline(), mark the cgroup dead and proceed with the rest of destruction. Once all cgroup references are gone, the cgroup is RCU-freed.

2 this source code (personal understandings)

This function implements Step1. After this step, @cgrp is gone as far as the userland is concerned and a new cgroup with the same name may be created. As

cgroup doesn't care about the names internally, this doesn't cause any problem.

4 Souce Code

```
static int cgroup_destroy_locked(struct cgroup *cgrp)
        __releases(&cgroup_mutex) __acquires(&cgroup_mutex)
{
// Definitions
        struct cgroup_subsys_state *css; // used by cgroup to manage subsys.
        struct cgrp_cset_link *link; //for describing the relation between css_set and cgroup.
        int ssid; //service set identifier
// Check for avoiding dead lock
        lockdep_assert_held(&cgroup_mutex);
// Only when migration can raise populated from zero and we're already holding cgroup_mutex.
// 'cgroup_is_populated' is used to check whether the @cgrp is populated by compared with '0'.
// 'EBUSY' stands for 'device or resource is busy error'.
        if (cgroup_is_populated(cgrp))
                return -EBUSY;
// Make sure there's no live children. We can't test emptiness of ->self.children as dead
// children linger on it while being drained; otherwise, "rmdir parent/child parent" may fail.
// 'css_has_online_children' is used to check whether there's live children.
        if (css_has_online_children(&cgrp->self))
                return -EBUSY;
// Mark @cgrp and the associated csets dead. The former prevents further task migration and
// child creation by disabling cgroup_lock_live_group(). The latter makes the csets ignored by
// the migration path.
        cgrp->self.flags &= ~CSS_ONLINE;
// Preparations done for the next operations to assure that the massacres will be processed
// in a normal pace and will not disturb the values.
        spin_lock_bh(&css_set_lock);
        list_for_each_entry(link, &cgrp->cset_links, cset_link)
                link->cset->dead = true;
        spin_unlock_bh(&css_set_lock);
// Initiate massacre of all css's by simply calling the 'kill_css'
        for_each_css(css, ssid, cgrp)
                kill_css(css);
// Remove @cgrp directory along with the base files. @cgrp has an extra ref on its kn.
        kernfs_remove(cgrp->kn);
// Check the parent of @cgrp
        check_for_release(cgroup_parent(cgrp));
// Put the base reference
        percpu_ref_kill(&cgrp->self.refcnt);
        return 0;
};
```