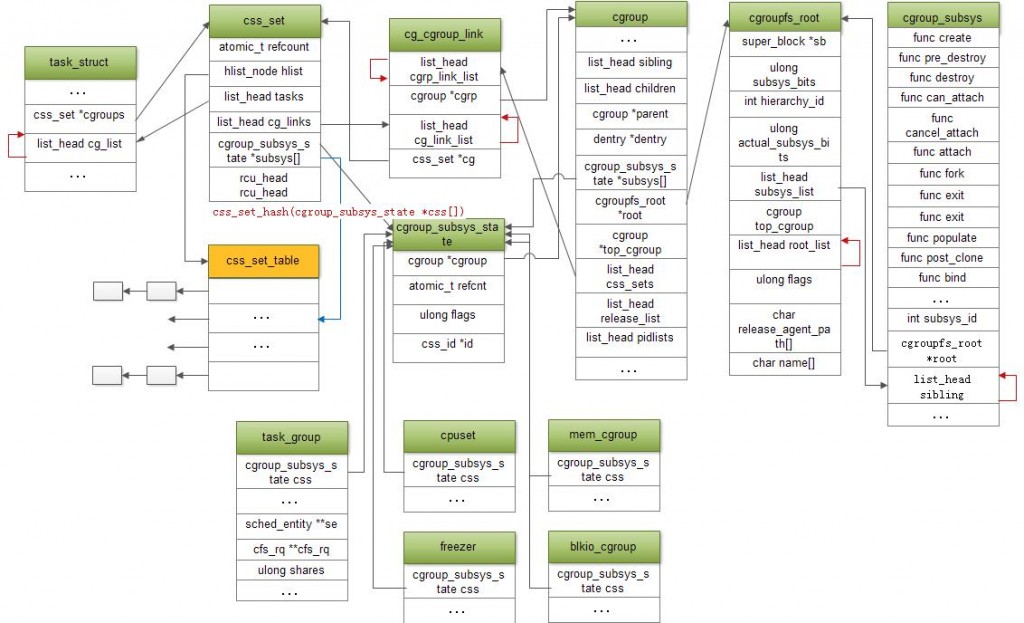
As the previous chapter mention there are many items in the cgroup system. How these items is describe in the kernel.



1. Root

Root is described by struct cgroupfs\_root

struct cgroupfs\_root {

……

/\* The bitmask of subsystems intended to be attached to this hierarchy \*/

unsigned long subsys\_mask;

/\* A list running through the attached subsystems \*/

struct list\_head subsys\_list;

/\* The default/top cgroup for this hierarchy \*/

struct cgroup top\_cgroup;

/\* A list running through the active hierarchies/Root \*/

struct list\_head root\_list;

/\* All cgroups on this root \*/

struct list\_head allcg\_list;

……

};

1. Subsystem

Subsystem is described by struct cgroup\_subsystem

struct cgroup\_subsys {

int subsys\_id; /\*id\*/

const char \*name; /\*name\*/

struct cgroupfs\_root \*root; /\*which Root this subsys belong to\*/

struct list\_head sibling; /\*link each subsystem in the same Root\*/

struct cftype \*base\_cftypes; /\*describe the subsystem files operations\*/

struct cftype\_set base\_cftset;

};

struct cftype {

char name[MAX\_CFTYPE\_NAME];

int (\*open)(struct inode \*inode, struct file \*file);

ssize\_t (\*read)(struct cgroup \*cgrp, struct cftype \*cft,

struct file \*file,

char \_\_user \*buf, size\_t nbytes, loff\_t \*ppos);

ssize\_t (\*write)(struct cgroup \*cgrp, struct cftype \*cft,

struct file \*file,

const char \_\_user \*buf, size\_t nbytes, loff\_t \*ppos);

}

In Linux kernel the subsystem is static defined, it contains the following subsystems.

#define SUBSYS(\_x) extern struct cgroup\_subsys \_x ## \_subsys;

static struct cgroup\_subsys \*subsys[CGROUP\_SUBSYS\_COUNT] = {

SUBSYS(cpuset) //struct cgroup\_subsys cpuset\_subsys

SUBSYS(debug) //struct cgroup\_subsys debug\_subsys

SUBSYS(cpu\_cgroup) //struct cgroup\_subsys cpu\_cgroup\_subsys

SUBSYS(cpuacct) //struct cgroup\_subsys cpuacct\_subsys

SUBSYS(mem\_cgroup) //struct cgroup\_subsys mem\_cgroup\_subsys

SUBSYS(devices) //struct cgroup\_subsys devices\_subsys

SUBSYS(freezer) //struct cgroup\_subsys freezer\_subsys

SUBSYS(net\_cls) //struct cgroup\_subsys net\_cls\_subsys

SUBSYS(blkio) // struct cgroup\_subsys blkio\_subsys

SUBSYS(perf) //struct cgroup\_subsys perf\_subsys

SUBSYS(net\_prio) //struct cgroup\_subsys net\_prio\_subsys

SUBSYS(hugetlb) //struct cgroup\_subsys hugetlb\_subsys

};

1. group

Group is described by struct cgroup

struct cgroup {

struct list\_head sibling; /\* my parent's children \*/

struct list\_head children; /\* my children \*/

struct cgroup \*parent; /\* my parent \*/

/\* Private pointers for each registered subsystem \*/

struct cgroup\_subsys\_state \*subsys[CGROUP\_SUBSYS\_COUNT];

struct cgroupfs\_root \*root; /\*which Root this group belong to\*/

struct cgroup \*top\_cgroup; /\*point to the default/top group in the Root\*/

/\* List of cg\_cgroup\_links pointing at css\_sets with tasks in this group , css\_set talked later\*/

struct list\_head css\_sets;

struct list\_head allcg\_node; /\* link to cgroupfs\_root->allcg\_list \*/

}

1. Tasks

A group contains several tasks/processes. A task only belongs to one of the group in a Root. In the default all tasks belong to the top group in this Root. But a task can belong to different groups in different Roots.

How a task associates with these groups?

task\_struct{

/\* A css\_set is a structure holding pointers to a set of cgroup\_subsys\_state objects \*/

struct css\_set \_\_rcu \*cgroups;

/\* cg\_list is link to css\_set->tasks \*/

struct list\_head cg\_list;

};

Task don’t communicate with groups directly. It contains a css\_set structure and a list linked to the css\_set.

A task has only one css\_set. But a css\_set may belong to multiple tasks because tasks may have the same subsystem resources and belong to the same group in different Roots. These tasks matched above situation share the same css\_set. The css\_set contains subsys state object that can control each subsystem resources.

struct css\_set {

/ \* List running through all cgroup groups in the same hash slot \*/

struct hlist\_node hlist;

/\*List running through all tasks using this css\_set\*/

struct list\_head tasks;

/\*List of cg\_cgroup\_link objects on link chains from

cgroups referenced from this css\_set \*/

struct list\_head cg\_links;

struct cgroup\_subsys\_state \*subsys[CGROUP\_SUBSYS\_COUNT];

};

A css\_set may belong to multiple groups in different Root because a task maybe attach to different Root. Also a group may have multiple css\_set because a group attach by multiple task whose css\_set may different.

How do css\_set associate with group?

struct cg\_cgroup\_link {

/\*List running through cg\_cgroup\_links associated with a

cgroup, link to cgroup->css\_sets \*/

struct list\_head cgrp\_link\_list;

struct cgroup \*cgrp;

/\*List running through cg\_cgroup\_links pointing at a

single css\_set object, link to css\_set->cg\_links\*/

struct list\_head cg\_link\_list;

struct css\_set \*cg;

};

So user can get the tasks in a specific group as following. Get each cg\_cgroup\_link through traversing on cgroup-> css\_sets, then find the css\_set through cg\_cgroup\_link->cg, finally traverse each task through css\_set->tasks.

Also user can check which group and Root this task attach to as following. Find the css\_set through task->css\_set, then get each the cg\_cgroup\_link through traversing css\_set->cg\_links, finally get cgroup through cg\_cgroup\_link->cgrp.

1. Subsystem object

struct cgroup\_subsys\_state as the base structure to describe each subsystem object that contain the subsystem resources control and management objects .

struct cgroup\_subsys\_state {

/\*The cgroup that this subsystem is attached to. \*/

struct cgroup \*cgroup;

};

1. Cpuset

struct cpuset {

struct cgroup\_subsys\_state css;

cpumask\_var\_t cpus\_allowed; /\* CPUs allowed to tasks in cpuset \*/

nodemask\_t mems\_allowed; /\* Memory Nodes allowed to tasks \*/

};

static inline struct cpuset \*cgroup\_cs(struct cgroup \*cont)

{

return container\_of(cgroup\_subsys\_state(cont, cpuset\_subsys\_id),

struct cpuset, css);

}

static inline struct cpuset \*task\_cs(struct task\_struct \*task)

{

return container\_of(task\_subsys\_state(task, cpuset\_subsys\_id),

struct cpuset, css);

}

1. Memory

struct mem\_cgroup {

struct cgroup\_subsys\_state css;

/…

};

static inline

struct mem\_cgroup \*mem\_cgroup\_from\_css(struct cgroup\_subsys\_state \*s)

{

return container\_of(s, struct mem\_cgroup, css);

}

1. Cpuacct

struct cpuacct {

struct cgroup\_subsys\_state css;

/\* cpuusage holds pointer to a u64-type object on every cpu \*/

u64 \_\_percpu \*cpuusage;

struct kernel\_cpustat \_\_percpu \*cpustat;

};

static inline struct cpuacct \*cgroup\_ca(struct cgroup \*cgrp)

{

return container\_of(cgroup\_subsys\_state(cgrp, cpuacct\_subsys\_id),

struct cpuacct, css);

}

static inline struct cpuacct \*task\_ca(struct task\_struct \*tsk)

{

return container\_of(task\_subsys\_state(tsk, cpuacct\_subsys\_id),

struct cpuacct, css);

}

1. Devices

struct dev\_cgroup {

struct cgroup\_subsys\_state css;

struct list\_head exceptions;

enum {

DEVCG\_DEFAULT\_ALLOW,

DEVCG\_DEFAULT\_DENY,

} behavior;

};

static inline struct dev\_cgroup \*cgroup\_to\_devcgroup(struct cgroup \*cgroup)

{

return css\_to\_devcgroup(cgroup\_subsys\_state(cgroup, devices\_subsys\_id));

}

static inline struct dev\_cgroup \*task\_devcgroup(struct task\_struct \*task)

{

return css\_to\_devcgroup(task\_subsys\_state(task, devices\_subsys\_id));

}

1. freezer

enum freezer\_state {

CGROUP\_THAWED = 0,

CGROUP\_FREEZING,

CGROUP\_FROZEN,

};

struct freezer {

struct cgroup\_subsys\_state css;

enum freezer\_state state;

spinlock\_t lock; /\* protects \_writes\_ to state \*/

};

static inline struct freezer \*cgroup\_freezer(

struct cgroup \*cgroup)

{

return container\_of(

cgroup\_subsys\_state(cgroup, freezer\_subsys\_id),

struct freezer, css);

}

static inline struct freezer \*task\_freezer(struct task\_struct \*task)

{

return container\_of(task\_subsys\_state(task, freezer\_subsys\_id),

struct freezer, css);

}