cgroups

What is it?

In the Linux kernel, **Control Groups (cgroups)** extend the system by providing additional structures that allow tasks (i.e., processes and threads) to be associated with a specific set of resource management rules (such as CPU limits, memory limits, etc.).

Two principle components:

- A mechanism for **hierarchically grouping** process
- A set of **controllers** (subsystems, kernel components) that manage, control, or monitor processes in cgroups

Hierarchy

- cgroups are organized in a tree-like structure, where each node (cgroup) represents a set of processes.
- Hierarchies allow the creation of parent and child cgroups, where limits can be inherited.

Controllers

Each cgroup can control specific system resources through controllers:

- **CPU (cpu, cpuacct)**: Limit and track CPU usage.
- **Memory** (memory): Limit and monitor memory usage.
- **Block I/O (blkio)**: Control and monitor disk I/O.
- **Network (net_cls, net_prio)**: Manage network bandwidth and traffic priority.
- Others: **PIDs**, **devices**, **freezer,** etc.

A task can be part of **multiple cgroups**, but **only one cgroup per controller** (e.g., CPU, memory, I/O, etc.).

Internals

Tasks and css_set (cgroup subsystem state set):

- Each task (process or thread) in the system has a pointer to a css_set, which tracks its membership in different cgroups (one per controller).
- css_set is a structure that holds pointers to the resource states for each cgroup subsystem (CPU, memory, etc.).

Reference counting:

- The css_set is reference-counted. When tasks share the same cgroup membership, they share the same css_set.
- Reference counting ensures efficient memory management. When no tasks reference a css_set, it can be safely destroyed.

Changing cgroups:

- When a task moves to a new cgroup (e.g., from one memory cgroup to another), the kernel updates the task's css_set, creating a new one if needed, and adjusting reference counts.

```
struct task_struct {
	/* Other fields for process information */
	struct css_set *cgroups; // Points to the css_set of the task
};

struct css_set {
	struct list_head tasks; // List of tasks in this css_set
	struct cgroup_subsys_state *subsys[CGROUP_SUBSYS_COUNT]; // Array of subsys states
	atomic_t refcount; // Reference count for memory management
};
```

Example

Cgroups: A for CPU, X for Memory, Y for Memory

Task1 and Task2 are both in cgroup A and cgroup X
Move Task1 to Y cgroup

```
shared_css_set.refcount--;

// new css_set
struct css_set new_css_set_task1 = {
        .subsys[CPU_SUBSYS] = cgroup_A_cpu_subsys_state, // Same CPU cgroup (cgroup A)
        .subsys[MEMORY_SUBSYS] = cgroup_Y_mem_subsys_state, // New memory cgroup (cgroup Y)
        .refcount = 1, // New reference count for this css_set
};

task1.cgroups = &new_css_set_task1;
```

Virtual File System Interaction

- **cgroup filesystem (`cgroupfs`)**:
  - cgroups expose their control interfaces via a **virtual filesystem** mounted at `/sys/fs/cgroup/`.
  - Each controller (e.g., CPU, memory, blkio) has its own directory or file under `/sys/fs/cgroup/`, allowing direct interaction with the cgroup hierarchy and subsystems.

- **Key directories and files**:
  - **/sys/fs/cgroup/**:
    The top-level directory for cgroups, containing subdirectories for different controllers (e.g., `cpu`, `memory`, `blkio`).
  - Inside each controller's directory, there are files that represent control parameters (e.g., `cpu.shares`, `memory.limit_in_bytes`).

```
echo 512M > /sys/fs/cgroup/memory/my_cgroup/memory.limit_in_bytes
```

cgroups v1 vs. cgroups v2

cgroups v1:

- Multiple hierarchies, each for one or more controllers.
- Each controller operates independently, which can lead to inconsistencies in resource management.

cgroups v2:

- Unified hierarchy: All controllers must be attached to a single hierarchy.
- Simplified and more consistent resource control across subsystems.
- Provides better isolation, particularly for containerized environments (used by systemd and container runtimes like Docker).

Resources

https://www.man7.org/

The Linux Programming Interface - Michael Kerrisk