OpenEuler

cgroup介绍

目录

CONTENT

OpenEuler

cgroup简介

cgroup的概念

cgroup是Linux提供的一种根据一些特定的限制，将一组任务和起子任务以层级组的形式进行聚合/隔离的机制

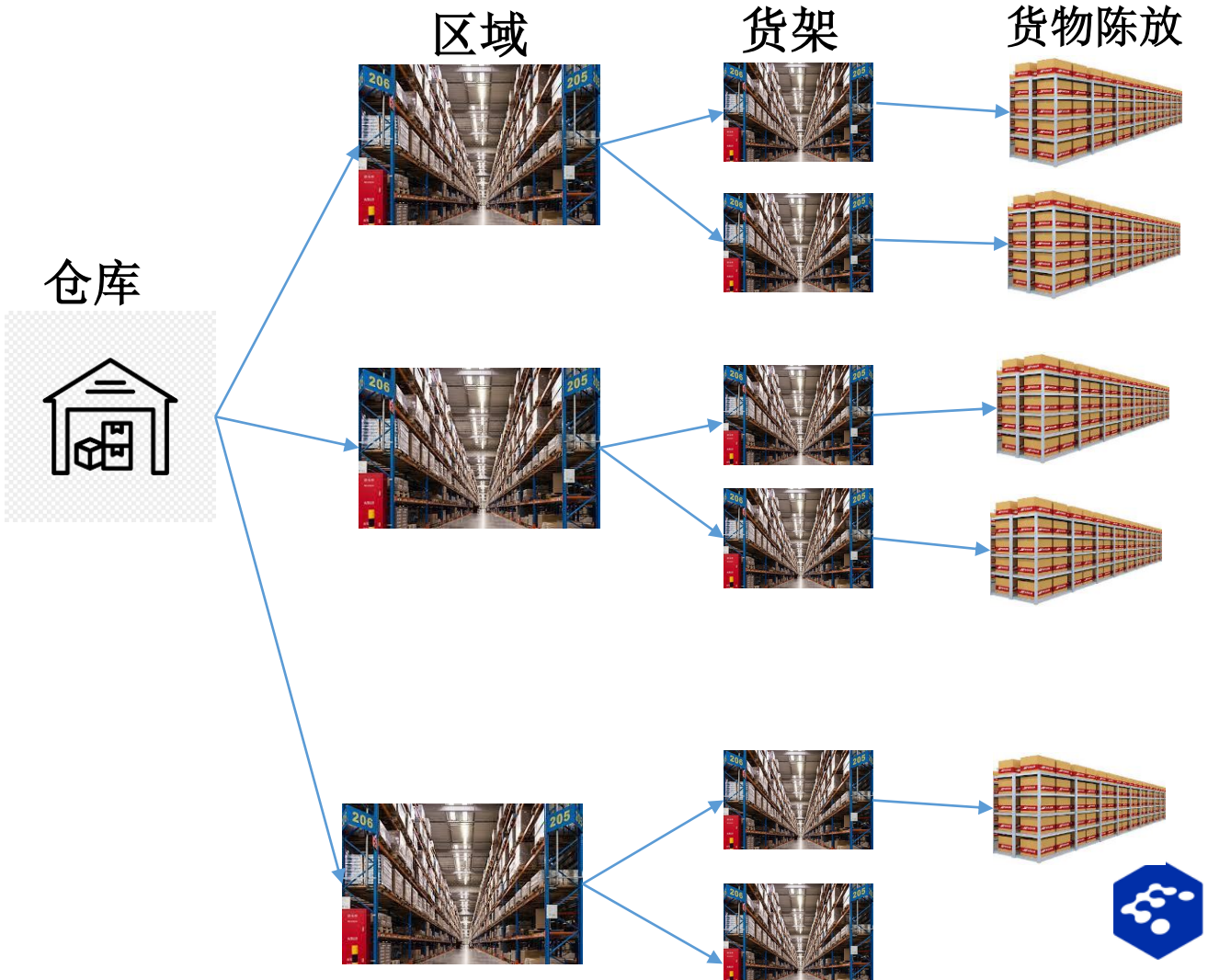
Control Groups provide a mechanism for **aggregating/partitioning** sets of tasks, and all their future children, into hierarchical groups with specialized behaviours

spcialized behaviours:

资源限制（memory/cpu/block io）

优先级控制（cpuset）

状态控制 （freezer）



OpenEuler

cgroup简介

相关概念介绍

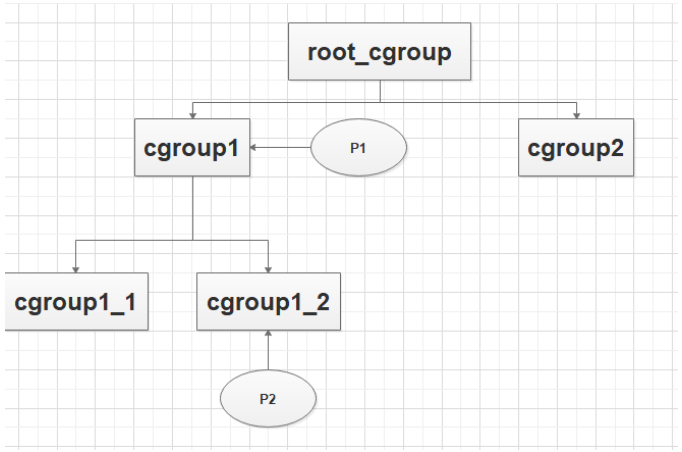
任务（task）：系统中的一个进程，在内核中的表现为struct task_struct

层级（hierarchy）：cgroup以树的形式组织，每一个树称之为一个层级

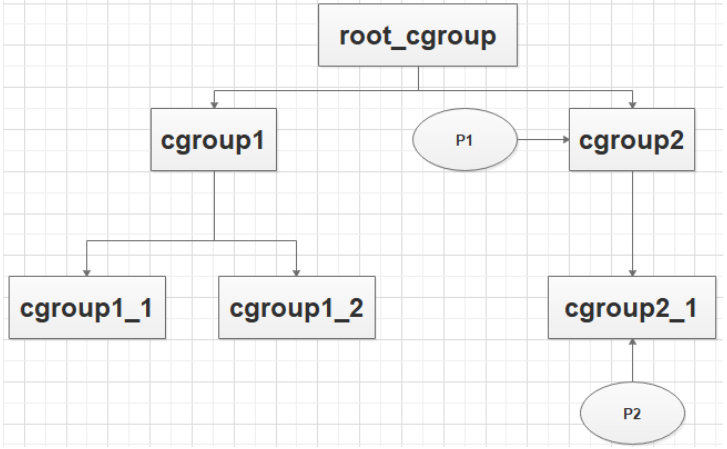
子系统（subsystem）：一个资源控制器

memory：设定内存使用限制，统计内存使用情况
HugeTLB：限制透明大页的使用量
cpu：限制进程的cpu使用率
cpuset：为进程分配单独的cpu/mem节点
blkio：为块设备设定输入/输出限制
devices：允许或者拒绝进程访问设备
net_cls：标记进程的网络数据包，并进行控制
net_prio：动态配置进程每隔网络接口的流量优先级
freezer：挂起或者恢复进程

cpu attach Hierarchy A



memory attach Hierarchy B



OpenEuler

cgroup简介

cgroup内核与用户态交互

文件系统

mount/umount

cgroupfs

```
struct file_system_type cgroup_fs_type = {
        .name           = "cgroup",
        .init_fs_context        = cgroup_init_fs_context,
        .parameters     = cgroup1_fs_parameters,
        .kill_sb        = cgroup_kill_sb,
        .fs_flags       = FS_USERNS_MOUNT,
};
```

目录操作

mkdir/rmdir

cgroup framework

```
static struct kernfs_syscall_ops cgroup_kf_syscall_ops = {
        .show_options   = cgroup_show_options,
        .mkdir          = cgroup_mkdir,
        .rmdir          = cgroup_rmdir,
        .show_path      = cgroup_show_path,
};
```

进程迁移

echo pid > xxx/cgroup.procs

cgroupfs

cgroup framework

subsystem

资源限制

echo xxx > xx/..limit

cgroupfs

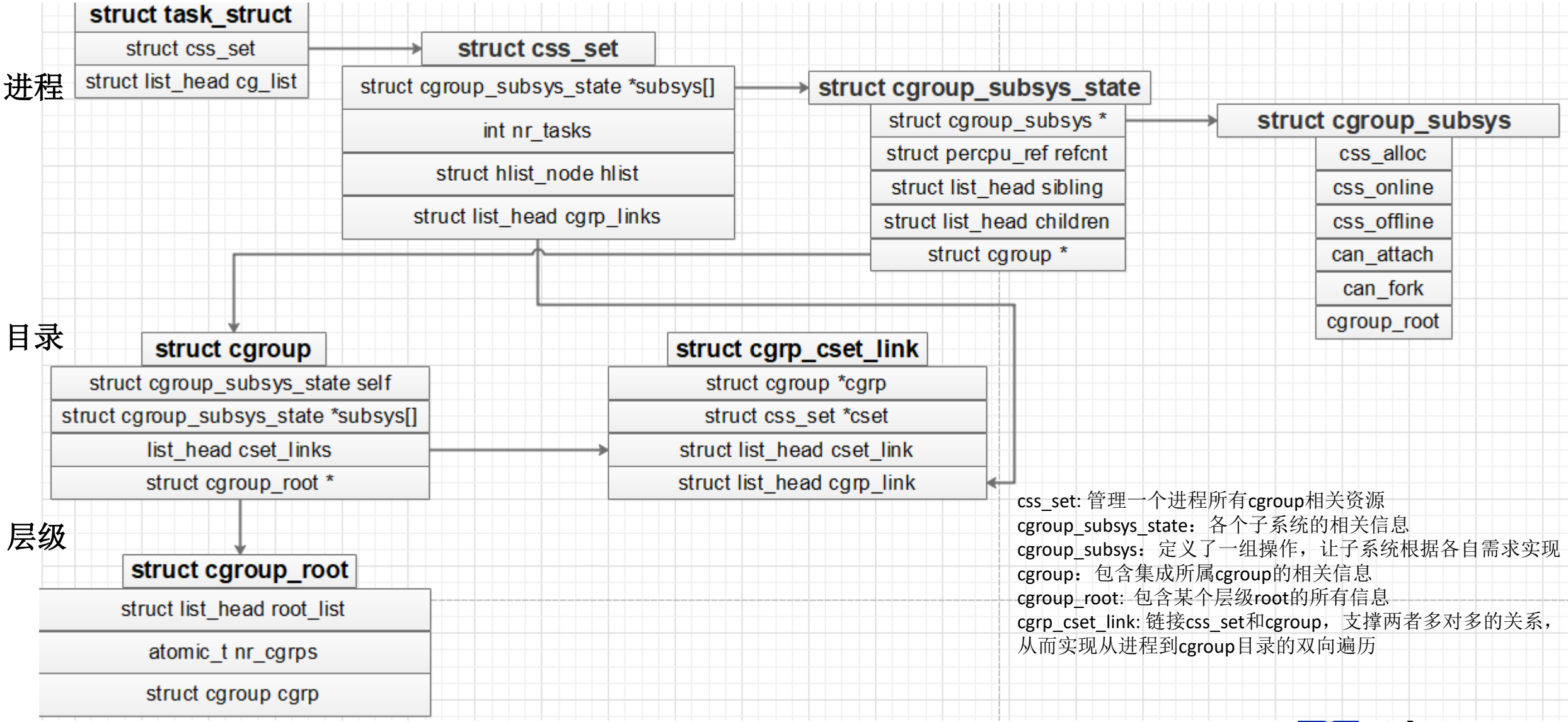
subsystem cftype

```
static struct cftype mem_cgroup_legacy_files[] = {
        {
                .name = "usage_in_bytes",
                .private = MEMFILE_PRIVATE(_MEM, RES_USAGE),
                .read_u64 = mem_cgroup_read_u64,
        },
        {
                .name = "max_usage_in_bytes",
                .private = MEMFILE_PRIVATE(_MEM, RES_MAX_USAGE),
                .write = mem_cgroup_reset,
                .read_u64 = mem_cgroup_read_u64,
        },
        {
                .name = "limit_in_bytes",
                .private = MEMFILE_PRIVATE(_MEM, RES_LIMIT),
                .write = mem_cgroup_write,
                .read_u64 = mem_cgroup_read_u64,
        },
```

OpenEuler

cgroup简介

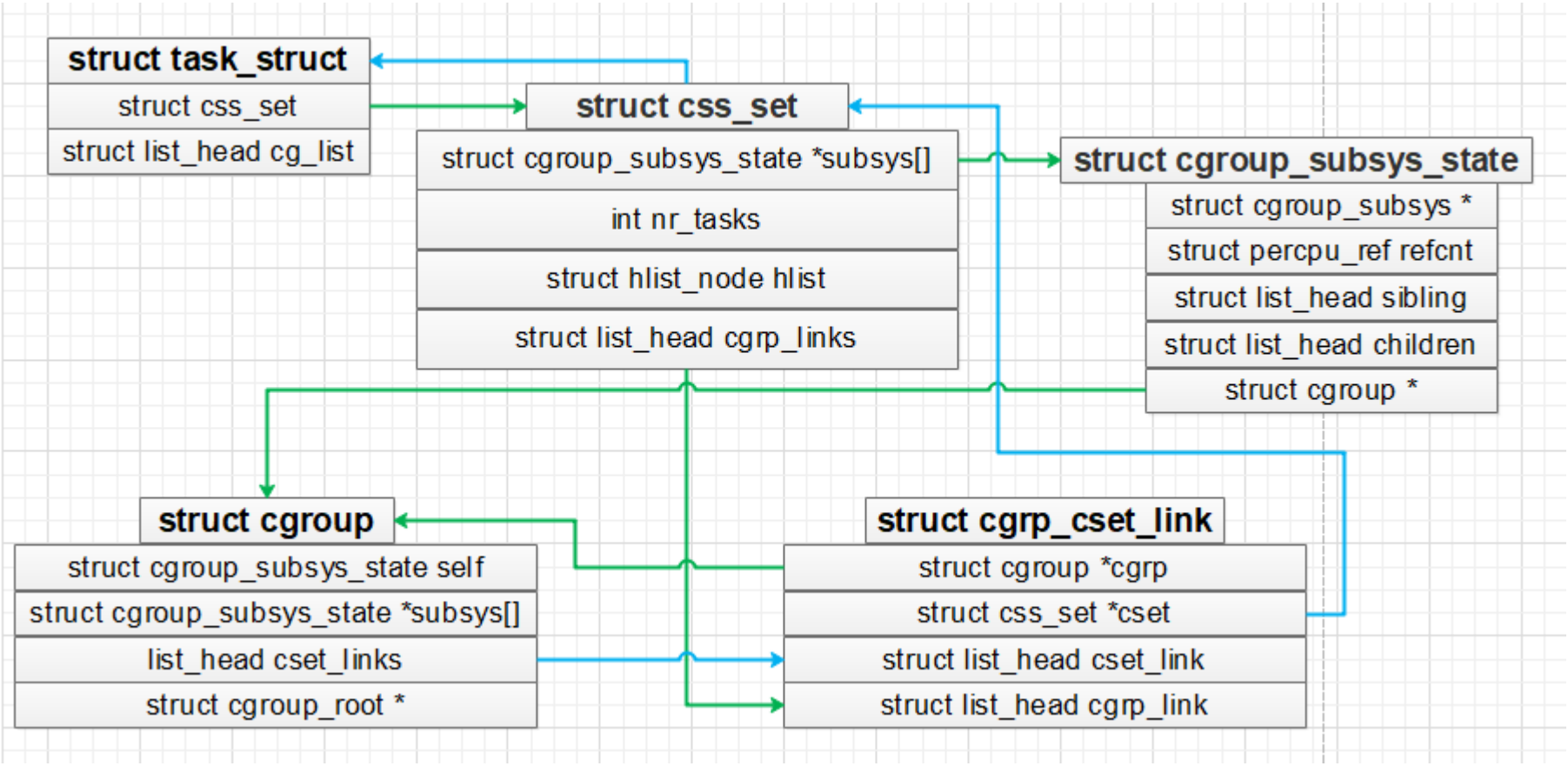
- cgroup主要结构体



css_set: 管理一个进程所有cgroup相关资源
cgroup_subsys_state：各个子系统的相关信息
cgroup_subsys：定义了一组操作，让子系统根据各自需求实现
cgroup：包含集成所属cgroup的相关信息
cgroup_root: 包含某个层级root的所有信息
cgrp_cset_link: 链接css_set和cgroup，支撑两者多对多的关系，从而实现从进程到cgroup目录的双向遍历

cgroup简介

- cgroup主要结构体



绿色箭头：task_struct->cgroup
蓝色箭头：cgroup->task

struct cgroup vs struct css_set (多对多)

one css_set -> different cgroup_subsys_state -> different cgroup

one cgroup -> different task -> different css_set

OpenEuler

cgroup演进

cgroupv1 vs cgroupv2

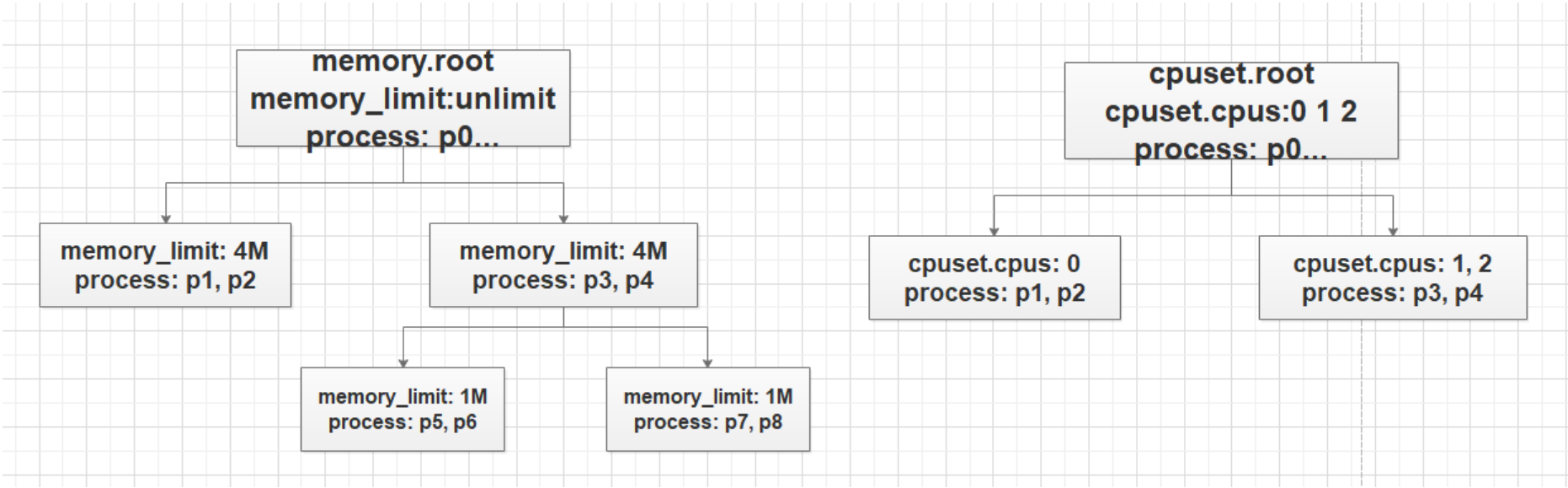
```
pm@ubuntu1804:~/API/linux$ cd /sys/fs/cgroup
pm@ubuntu1804:/sys/fs/cgroup$ ls
blkio  cpuacct      cpuset   freezer  memory   net_cls,net_prio  perf_event  rdma     unified
cpu    cpu,cpuacct  devices  hugetlb  net_cls  net_prio          pids        systemd
pm@ubuntu1804:/sys/fs/cgroup$
pm@ubuntu1804:/sys/fs/cgroup$ cd memory
pm@ubuntu1804:/sys/fs/cgroup/memory$ ls
cgroup.clone_children  memory.kmem.limit_in_bytes          memory.kmem.usage_in_bytes       memory.soft_limit_in_bytes  system.slice
cgroup.event_control   memory.kmem.max_usage_in_bytes      memory.limit_in_bytes            memory.stat                 tasks
cgroup.procs           memory.kmem.slabinfo                memory.max_usage_in_bytes        memory.swappiness           user.slice
cgroup.sane_behavior   memory.kmem.tcp.failcnt             memory.move_charge_at_immigrate  memory.usage_in_bytes
memory.failcnt         memory.kmem.tcp.limit_in_bytes      memory.numa_stat                 memory.use_hierarchy
memory.force_empty     memory.kmem.tcp.max_usage_in_bytes  memory.oom_control               notify_on_release
memory.kmem.failcnt    memory.kmem.tcp.usage_in_bytes      memory.pressure_level            release_agent
```

- cgroupv1

支持的子系统：cpu, cpuset, memory, devices, freezer, net_cls, net_prio, blkio, perf_event, hugetlb, pids, rdma, misc, cpuacct

合入版本： Linux2.6开始

管理方式：多个层级分开管理，每个子cgroup同时管理资源和进程二者



OpenEuler

cgroup演进

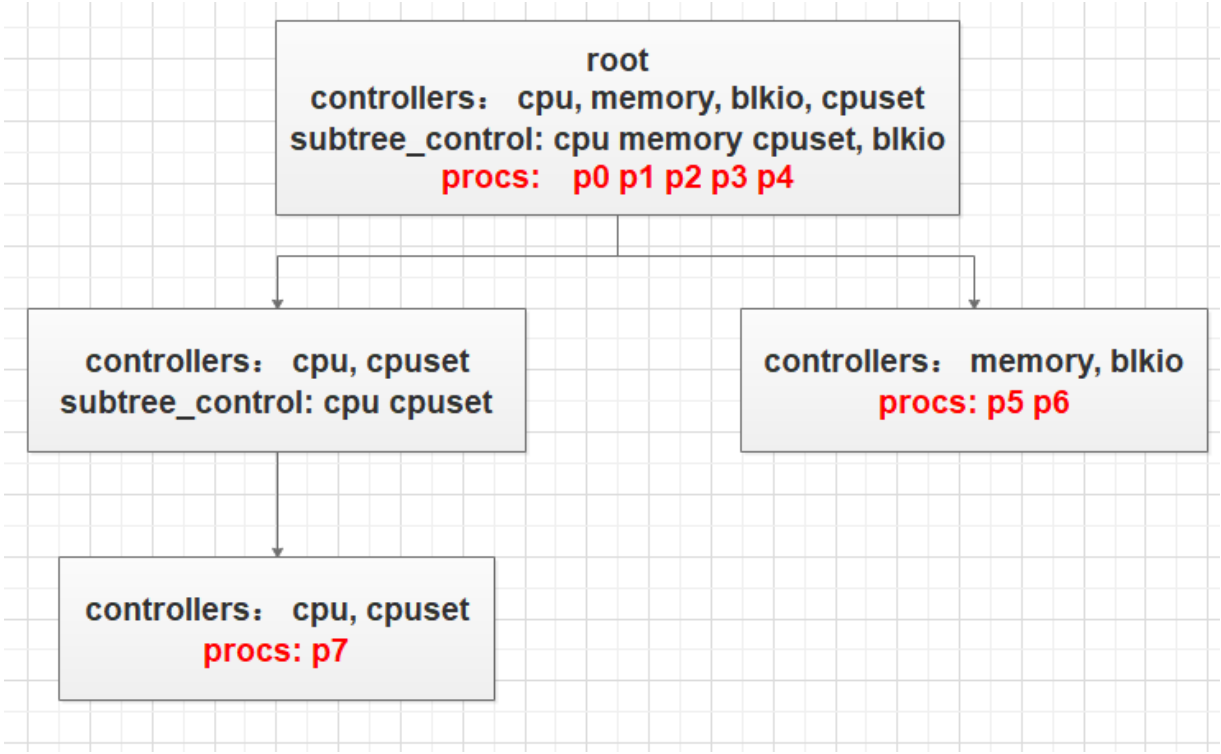
cgroupv1 vs cgroupv2

- cgroupv2

支持的子系统：不支持cpuacct net_cls 和net_prio

合入版本： Linux4.5开始

管理方式：统一层级管理，资源和进程管理分开



OpenEuler

cgroupv1 vs cgroupv2

- cgroupv1与cgroupv2的异同

cgroupv1	cgroupv2
多层级设计导致进程管理较为混乱	采用单一层级，cgroup代表进程分组，子系统用于资源控制
各个子系统之间很难协同合作（writeback功能）	挂载统一层级，实现多系统协同工作的功能
Memcg资源限制手段简单，实际业务无法很好平衡	采用了分级管理的调节手段,可以提高性能，解决临时内存峰值的情况
不支持rootless	支持rootless
不支持PSI功能	支持PSI功能

cgroup演进

容器运行时	支持版本	版本发行时间	备注
containered	v1.4	2020/8/17	
runc	V1.0-rc93	2021/2/4	从v1.0-rc91开始支持
LXC	3.0.4	2019/1	从3.0.0开始支持
Podman	2.2.1	2020/12/9	
cAdvisor		2019/9/6	
Docker	20.10	2020/12	Rootless Container正式可用，允许Docker daemon在none-root用户状态运行

cgroupv2更适合将逐步并取代cgroupv1

OpenEuler

memcg子系统

- 引进内存子系统的原因：将系统中一些进程的内存资源通过某种限制进行隔离，从而导致不同的内存处理行为；

- memcg的特点：

1. 可以统计匿名页，文件页，swap等的使用量，并且对一个cgroup中这些类别的内存上限进行限制；
2. Pages映射到的是对应每一个memcg的LRU链表；
3. 迁移任务时可以根据业务需要选择是否将迁移前任务的内存统计到新的memcg中；
4. 可以对一些行为，例如：触发memcg级别oom，内存压力过大等统计发生的次数；
5. 实际业务中，root memcg的limit是不做限制的；

```
cgroup.clone_children       memory.kmem.max_usage_in_bytes      memory.max_usage_in_bytes          memory.pressure_level       release_agent
cgroup.event_control        memory.kmem.slabinfo                memory.memsw.failcnt               memory.qos_level            system.slice
cgroup.procs                memory.kmem.tcp.failcnt             memory.memsw.limit_in_bytes        memory.soft_limit_in_bytes  tasks
cgroup.sane_behavior        memory.kmem.tcp.limit_in_bytes      memory.memsw.max_usage_in_bytes    memory.stat                 user.slice
memory.failcnt              memory.kmem.tcp.max_usage_in_bytes  memory.memsw.usage_in_bytes        memory.swappiness
memory.force_empty          memory.kmem.tcp.usage_in_bytes      memory.move_charge_at_immigrate    memory.usage_in_bytes
memory.kmem.failcnt         memory.kmem.usage_in_bytes          memory.numa_stat                   memory.use_hierarchy
memory.kmem.limit_in_bytes  memory.limit_in_bytes               memory.oom_control                 notify_on_release
```

memcg子系统

- **挂载/卸载memcg子系统操作：**

mount –t cgroup –o memory [name] [dir]

umount [dir]

```
root@ubuntu:~# mount -t cgroup -o memory memcg /tmp/test
root@ubuntu:~# mount | grep cgroup
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,mode=755)
cgroup on /sys/fs/cgroup/unified type cgroup2 (rw,nosuid,nodev,noexec,relatime)
cgroup on /sys/fs/cgroup/systemd type cgroup (rw,nosuid,nodev,noexec,relatime,xattr,name=systemd)
cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup (rw,nosuid,nodev,noexec,relatime,net_cls,net_prio)
cgroup on /sys/fs/cgroup/blkio type cgroup (rw,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/hugetlb type cgroup (rw,nosuid,nodev,noexec,relatime,hugetlb)
cgroup on /sys/fs/cgroup/rdma type cgroup (rw,nosuid,nodev,noexec,relatime,rdma)
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,cpu,cpuacct)
cgroup on /sys/fs/cgroup/perf_event type cgroup (rw,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/freezer type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/devices type cgroup (rw,nosuid,nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
memcg on /tmp/test type cgroup (rw,relatime,memory)
root@ubuntu:~# umount /tmp/test
root@ubuntu:~# mount | grep cgroup
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,mode=755)
cgroup on /sys/fs/cgroup/unified type cgroup2 (rw,nosuid,nodev,noexec,relatime)
cgroup on /sys/fs/cgroup/systemd type cgroup (rw,nosuid,nodev,noexec,relatime,xattr,name=systemd)
cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup (rw,nosuid,nodev,noexec,relatime,net_cls,net_prio)
cgroup on /sys/fs/cgroup/blkio type cgroup (rw,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/hugetlb type cgroup (rw,nosuid,nodev,noexec,relatime,hugetlb)
cgroup on /sys/fs/cgroup/rdma type cgroup (rw,nosuid,nodev,noexec,relatime,rdma)
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,cpu,cpuacct)
cgroup on /sys/fs/cgroup/perf_event type cgroup (rw,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/freezer type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/devices type cgroup (rw,nosuid,nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
```

memcg子系统

- **通过memcg的内存限制对memcg中的进程进行隔离**

1. 对memcg进行内存上限的限制：

echo xxx > /[memcg_dir]/memory.limit_in_bytes

cat /[memcg_dir]/memory_limit_in_bytes

2.迁移一个进程到指定的memcg：

echo [pid] > /[memcg_dir]/cgroup.procs

```
/tmp/test/test # echo 2M > memory.limit_in_bytes
/tmp/test/test # echo $$ > cgroup.procs
/tmp/test/test # cd
/ # cd test
/test # ls
kasan                       test
kpatch-0001-test-random.ko  test.sh
ljl.sh                      test_memcg.sh
memcg_malloc                tmp
memcg_testcase_malloc
/test # ./test &
/test # [  137.674317] test invoked oom-killer: gfp_mask=0xcc0(GFP_KERNEL), order=0, oom_score_adj=0
[  137.675041] CPU: 2 PID: 138 Comm: test Not tainted 5.10.0+ #7
[  137.675276] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS rel-1.12.1-0-ga5cab58e9a3f-prebuilt.qemu.org 04/01/2014
[  137.675737] Call Trace:
[  137.675737]  dump_stack+0x57/0x6a
[  137.675737]  dump_header+0x4a/0x1e7
[  137.675737]  oom_kill_process.cold.9+0xb/0x10
[  137.675737]  out_of_memory+0x1bd/0x540
[  137.675737]  mem_cgroup_out_of_memory+0xe8/0x100
[  137.675737]  try_charge+0x6f5/0x740
```

memcg子系统

- 获取当前memcg的内存使用量和某些类别内存的使用量

1. 查看某个memcg的内存使用量：

cat [memcg_dir]/memory.usage_in_bytes

```
/tmp/test/test # cat memory.usage_in_bytes
17117184
```

2. 查看某个memcg中某种内存的使用量：

```
/tmp/test/test # cat memory.stat
cache 0
rss 16920576
rss_huge 2097152
shmem 0
mapped_file 0
dirty 0
writeback 0
swap 0
pgpgin 4174
pgpgout 554
pgfault 4299
pgmajfault 0
inactive_anon 16867328
active_anon 8192
inactive_file 0
active_file 0
unevictable 0
```

sta

```
total_cache 0
total_rss 16920576
total_rss_huge 2097152
total_shmem 0
total_mapped_file 0
total_dirty 0
total_writeback 0
total_swap 0
total_pgpgin 4174
total_pgpgout 554
total_pgfault 4299
total_pgmajfault 0
total_inactive_anon 16867328
total_active_anon 8192
total_inactive_file 0
total_active_file 0
total_unevictable 0
```

```
hierarchical_memory_limit 9223372036854771712
hierarchical_memsw_limit 9223372036854771712
```

OpenEuler

openEuler有关cgroup的增强

1）使能了files cgroup，实现了对进程组打开文件数进行控制，从而实现了进程隔离；

2）实现memcg qos

- 在发生OOM时，优先kill掉offline的memcg中的进程，保障online业务的运行

- 将cgroupv2实现的内存分类在cgroupv1中实现，从而减弱兄弟cgroup之间的性能影响，解决某些任务出现内存短时间高峰的问题；

- 实现memcg级别的后台回收，使得可以及时回收memcg的dirty内存，解决某些任务短时间内产生大量dirty内存但是没有达到全局的回收dirty标准，从而使任务出现问题；

3）实现cpu qos

对在线任务和离线任务进行分组，调度器根据分组，从而优先选择online任务，保障online业务优先运行

OpenEuler

✓ openEuler kernel gitee 仓库

源代码仓库

https://gitee.com/openeuler/kernel

欢迎大家多多 Star，多多参与社区开发，多多贡献补丁。

✓ maillist、issue、bugzilla

可以通过邮件列表、issue、bugzilla 参与社区讨论

欢迎大家多多讨论问题，发现问题多提 issue、bugzilla

https://gitee.com/openeuler/kernel/issues

https://bugzilla.openeuler.org

kernel@openeuler.org

✓ openEuler kernel SIG 微信技术交流群

请扫描右方二维码添加小助手微信

或者直接添加小助手微信（微信号：openeuler-kernel）

备注“交流群”或“技术交流”

加入 openEuler kernel SIG 技术交流群

技术交流

OpenEuler

Thank you

OpenEuler