

Hi Daisuke,

Most of my comments below are to do with style issues with cgroups, rather than the details of the memory management code.

2008/3/4 Daisuke Nishimura <nishimura@mxp.nes.nec.co.jp>:

```
> +/*
> + * A page_cgroup page is associated with every page descriptor. The
> + * page_cgroup helps us identify information about the cgroup
> + */
> +struct page_cgroup {
> +    struct list_head lru;      /* per cgroup LRU list */
> +    struct page *page;
> +    struct mem_cgroup *mem_cgroup;
> +#ifdef CONFIG_CGROUP_SWAP_LIMIT
> +    struct mm_struct *pc_mm;
> +#endif
> +    atomic_t ref_cnt;          /* Helpful when pages move b/w */
> +                                /* mapped and cached states */
> +    int    flags;
> +};
>
> +
> +#ifdef CONFIG_CGROUP_SWAP_LIMIT
> +struct swap_cgroup {
> +    struct cgroup_subsys_state css;
> +    struct res_counter res;
> +};
> +
> +static inline struct swap_cgroup *swap_cgroup_from_cgrp(struct cgroup *cgrp)
> +{
> +    return container_of(cgroup_subsys_state(cgrp, swap_subsys_id),
> +                        struct swap_cgroup,
> +                        css);
> +}
> +
> +static inline struct swap_cgroup *swap_cgroup_from_task(struct task_struct *p)
> +{
> +    return container_of(task_subsys_state(p, swap_subsys_id),
> +                        struct swap_cgroup, css);
> +}
```

Can't these definitions be moved into swap_limit.c?

```

> @@ -254,15 +243,27 @@ struct mem_cgroup *mem_cgroup_from_task(
> void mm_init_cgroup(struct mm_struct *mm, struct task_struct *p)
> {
>     struct mem_cgroup *mem;
> #ifdef CONFIG_CGROUP_SWAP_LIMIT
> +     struct swap_cgroup *swap;
> #endif
>
>     mem = mem_cgroup_from_task(p);
>     css_get(&mem->css);
>     mm->mem_cgroup = mem;
> +
> #ifdef CONFIG_CGROUP_SWAP_LIMIT
> +     swap = swap_cgroup_from_task(p);
> +     css_get(&swap->css);
> +     mm->swap_cgroup = swap;
> #endif

```

My feeling is that it would be cleaner to move this code into swap_limit.c, and have a separate mm_init_swap_cgroup() function. (And a mm_free_swap_cgroup() function).

```

> +     pc = page_get_page_cgroup(page);
> +     if (WARN_ON(!pc))
> +         mm = &init_mm;
> +     else
> +         mm = pc->pc_mm;
> +     BUG_ON(!mm);

```

Is this safe against races with the mem.force_empty operation?

```

> +
> +     rcu_read_lock();
> +     swap = rcu_dereference(mm->swap_cgroup);
> +     rcu_read_unlock();
> +     BUG_ON(!swap);

```

Is it safe to do rcu_read_unlock() while you are still planning to operate on the value of "swap"?

```

> +
> +static ssize_t swap_cgroup_read(struct cgroup *cgrp,
> +                                struct cftype *cft, struct file *file,
> +                                char __user *userbuf, size_t nbytes,
> +                                loff_t *ppos)
> +{
> +     return res_counter_read(&swap_cgroup_from_cgrp(cgrp)->res,
> +                             cft->private, userbuf, nbytes, ppos,

```

```
> +                NULL);
> +}
```

Can you use the cgroups read_u64 method, and just call res_counter_read_u64?

```
> +
> +static int swap_cgroup_write_strategy(char *buf, unsigned long long *tmp)
> +{
> +    *tmp = memparse(buf, &buf);
> +    if (*buf != '\0')
> +        return -EINVAL;
> +
> +    /*
> +     * Round up the value to the closest page size
> +     */
> +    *tmp = ((*tmp + PAGE_SIZE - 1) >> PAGE_SHIFT) << PAGE_SHIFT;
> +    return 0;
> +}
```

This is the same as mem_cgroup_write_strategy. As part of your patch, can you create a res_counter_write_pagealign() strategy function in res_counter.c and use it from the memory and swap cgroups?

```
> +
> +#ifdef CONFIG_CGROUP_SWAP_LIMIT
> +    p->swap_cgroup = vmalloc(maxpages * sizeof(*swap_cgroup));
> +    if (!(p->swap_cgroup)) {
> +        error = -ENOMEM;
> +        goto bad_swap;
> +    }
> +    memset(p->swap_cgroup, 0, maxpages * sizeof(*swap_cgroup));
> +#endif
```

It would be nice to only allocate these the first time the swap cgroup subsystem becomes active, to avoid the overhead for people not using it; even better if you can free it again if the swap subsystem becomes inactive again.

Paul

Containers mailing list
Containers@lists.linux-foundation.org
<https://lists.linux-foundation.org/mailman/listinfo/containers>
