

---

Subject: Re: [PATCH 4/5] Add fair-user scheduler

Posted by [roel](#) on Mon, 24 Sep 2007 23:39:39 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Srivatsa Vaddagiri wrote:

```
> Enable user-id based fair group scheduling. This is usefull for anyone
> who wants to test the group scheduler w/o having to enable
> CONFIG_CGROUPS.
>
> A separate scheduling group (i.e struct task_grp) is automatically created for
> every new user added to the system. Upon uid change for a task, it is made to
> move to the corresponding scheduling group.
>
> A /proc tunable (/proc/root_user_share) is also provided to tune root
> user's quota of cpu bandwidth.
>
> Signed-off-by : Srivatsa Vaddagiri <vatsa@linux.vnet.ibm.com>
> Signed-off-by : Dhaval Giani <dhaval@linux.vnet.ibm.com>
>
>
> ---
> include/linux/sched.h | 4 +++
> init/Kconfig          | 13 ++++++++
> kernel/sched.c        | 9 ++++++
> kernel/sched_debug.c | 52 ++++++++++++++++++++++++++++++++++++++
> kernel/user.c         | 43 ++++++++++++++++++++++++++++++++++++++
> 5 files changed, 121 insertions(+)
>
> Index: linux-2.6.23-rc6/include/linux/sched.h
> =====
> --- linux-2.6.23-rc6.orig/include/linux/sched.h
> +++ linux-2.6.23-rc6/include/linux/sched.h
> @@ -596,6 +596,10 @@ struct user_struct {
>  /* Hash table maintenance information */
>  struct hlist_node uidhash_node;
>  uid_t uid;
> +
> + #ifdef CONFIG_FAIR_USER_SCHED
> + struct task_grp *tg;
> + #endif
> };
>
> extern struct user_struct *find_user(uid_t);
> Index: linux-2.6.23-rc6/init/Kconfig
> =====
> --- linux-2.6.23-rc6.orig/init/Kconfig
> +++ linux-2.6.23-rc6/init/Kconfig
> @@ -289,6 +289,19 @@ config FAIR_GROUP_SCHED
```

```

> This feature lets cpu scheduler recognize task groups and control cpu
> bandwidth allocation to such task groups.
>
> +choice
> + depends on FAIR_GROUP_SCHED
> + prompt "Basis for grouping tasks"
> + default FAIR_USER_SCHED
> +
> + config FAIR_USER_SCHED
> + bool "user id"
> + help
> + This option will choose userid as the basis for grouping
> + tasks, thus providing equal cpu bandwidth to each user.
> +
> +endchoice
> +
> config SYSFS_DEPRECATED
> bool "Create deprecated sysfs files"
> default y
> Index: linux-2.6.23-rc6/kernel/sched.c
> =====
> --- linux-2.6.23-rc6.orig/kernel/sched.c
> +++ linux-2.6.23-rc6/kernel/sched.c
> @@ -199,7 +199,12 @@ struct task_grp init_task_grp = {
>     .cfs_rq = init_cfs_rq_p,
>     };
>
>
> +#ifdef CONFIG_FAIR_USER_SCHED
> +#define INIT_TASK_GRP_LOAD 2*NICE_0_LOAD
> +#else
> +#define INIT_TASK_GRP_LOAD NICE_0_LOAD
> +#endif
> +
> static int init_task_grp_load = INIT_TASK_GRP_LOAD;
>
> /* return group to which a task belongs */
> @@ -207,7 +212,11 @@ static inline struct task_grp *task_grp(
> {
>     struct task_grp *tg;
>
> +#ifdef CONFIG_FAIR_USER_SCHED
> + tg = p->user->tg;
> +#else
>     tg = &init_task_grp;
> +#endif
>
>     return tg;
> }

```

```

> Index: linux-2.6.23-rc6/kernel/sched_debug.c
> =====
> --- linux-2.6.23-rc6.orig/kernel/sched_debug.c
> +++ linux-2.6.23-rc6/kernel/sched_debug.c
> @@ -214,6 +214,49 @@ static void sysrq_sched_debug_show(void)
> sched_debug_show(NULL, NULL);
> }
>
> +#ifdef CONFIG_FAIR_USER_SCHED
> +
> +static DEFINE_MUTEX(root_user_share_mutex);
> +
> +static int
> +root_user_share_read_proc(char *page, char **start, off_t off, int count,
> + int *eof, void *data)
> +{
> + int len;
> +
> + len = sprintf(page, "%d\n", init_task_grp_load);
> +
> + return len;
> +}

```

or use this oneliner:

```

return sprintf(page, "%d\n", init_task_grp_load);

> +
> +static int
> +root_user_share_write_proc(struct file *file, const char __user *buffer,
> + unsigned long count, void *data)
> +{
> + unsigned long shares;
> + char kbuf[sizeof(unsigned long)+1];
> + int rc = 0;
> +
> + if (copy_from_user(kbuf, buffer, sizeof(kbuf)))
> + return -EFAULT;
> +
> + shares = simple_strtoul(kbuf, NULL, 0);
> +
> + if (!shares)
> + shares = NICE_0_LOAD;
> +
> + mutex_lock(&root_user_share_mutex);
> +
> + init_task_grp_load = shares;
> + rc = sched_group_set_shares(&init_task_grp, shares);

```

```

> +
> + mutex_unlock(&root_user_share_mutex);
> +
> + return (rc < 0 ? rc : count);
> +}
> +
> +#endif /* CONFIG_FAIR_USER_SCHED */
> +
> static int sched_debug_open(struct inode *inode, struct file *filp)
> {
>     return single_open(filp, sched_debug_show, NULL);
> @@ -236,6 +279,15 @@ static int __init init_sched_debug_procfs
>
>     pe->proc_fops = &sched_debug_fops;
>
> +#ifdef CONFIG_FAIR_USER_SCHED
> + pe = create_proc_entry("root_user_share", 0644, NULL);
> + if (!pe)
> +     return -ENOMEM;
> +
> + pe->read_proc = root_user_share_read_proc;
> + pe->write_proc = root_user_share_write_proc;
> +#endif
> +
>     return 0;
> }
>
> Index: linux-2.6.23-rc6/kernel/user.c
> =====
> --- linux-2.6.23-rc6.orig/kernel/user.c
> +++ linux-2.6.23-rc6/kernel/user.c
> @@ -50,8 +50,41 @@ struct user_struct root_user = {
>     .uid_keyring = &root_user_keyring,
>     .session_keyring = &root_session_keyring,
> #endif
> +#ifdef CONFIG_FAIR_USER_SCHED
> + .tg = &init_task_grp,
> +#endif
> };
>
> +#ifdef CONFIG_FAIR_USER_SCHED
> +static void sched_destroy_user(struct user_struct *up)
> +{
> + sched_destroy_group(up->tg);
> +}
> +
> +static int sched_create_user(struct user_struct *up)
> +{

```

```

> + int rc = 0;
> +
> + up->tg = sched_create_group();
> + if (IS_ERR(up->tg))
> + rc = -ENOMEM;
> +
> + return rc;
> +}
> +
> +static void sched_switch_user(struct task_struct *p)
> +{
> + sched_move_task(p);
> +}
> +
> +#else /* CONFIG_FAIR_USER_SCHED */
> +
> +static void sched_destroy_user(struct user_struct *up) { }
> +static int sched_create_user(struct user_struct *up) { return 0; }
> +static void sched_switch_user(struct task_struct *p) { }
> +
> +#endif /* CONFIG_FAIR_USER_SCHED */
> +
> /*
>  * These routines must be called with the uidhash spinlock held!
>  */
> @@ -109,6 +142,7 @@ void free_uid(struct user_struct *up)
> if (atomic_dec_and_lock(&up->__count, &uidhash_lock)) {
> uid_hash_remove(up);
> spin_unlock_irqrestore(&uidhash_lock, flags);
> + sched_destroy_user(up);
> key_put(up->uid_keyring);
> key_put(up->session_keyring);
> kmem_cache_free(uid_cachep, up);
> @@ -150,6 +184,13 @@ struct user_struct * alloc_uid(struct us
> return NULL;
> }
>
> + if (sched_create_user(new) < 0) {
> + key_put(new->uid_keyring);
> + key_put(new->session_keyring);
> + kmem_cache_free(uid_cachep, new);
> + return NULL;
> + }
> +
> /*
>  * Before adding this, check whether we raced
>  * on adding the same user already..
> @@ -157,6 +198,7 @@ struct user_struct * alloc_uid(struct us

```

```
> spin_lock_irq(&uidhash_lock);
> up = uid_hash_find(uid, hashent);
> if (up) {
> + sched_destroy_user(new);
> key_put(new->uid_keyring);
> key_put(new->session_keyring);
> kmem_cache_free(uid_cachep, new);
> @@ -184,6 +226,7 @@ void switch_uid(struct user_struct *new_
> atomic_dec(&old_user->processes);
> switch_uid_keyring(new_user);
> current->user = new_user;
> + sched_switch_user(current);
>
> /*
> * We need to synchronize with __sigqueue_alloc()
>
```

---

Containers mailing list

Containers@lists.linux-foundation.org

<https://lists.linux-foundation.org/mailman/listinfo/containers>

---