Topics in Operating Systems

Advanced isolation: Namespace and Cgroups

Debadatta Mishra, CSE, IITK

Recap: Linux kernel enablers for containers

1

Namespaces

Kernel abstraction for restricted system view

2

Control groups (cgroups)

Techniques for resource isolation (quota, usage etc.)

API

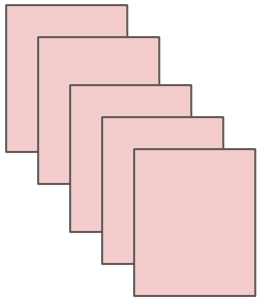
System calls: clone( ), unshare( ) and setns( )

API

Kernel sysfs API (/sys/fs/cgroup)

Container management frameworks (e.g., LXC, docker)

Containers



Namespaces in kernel: data structures

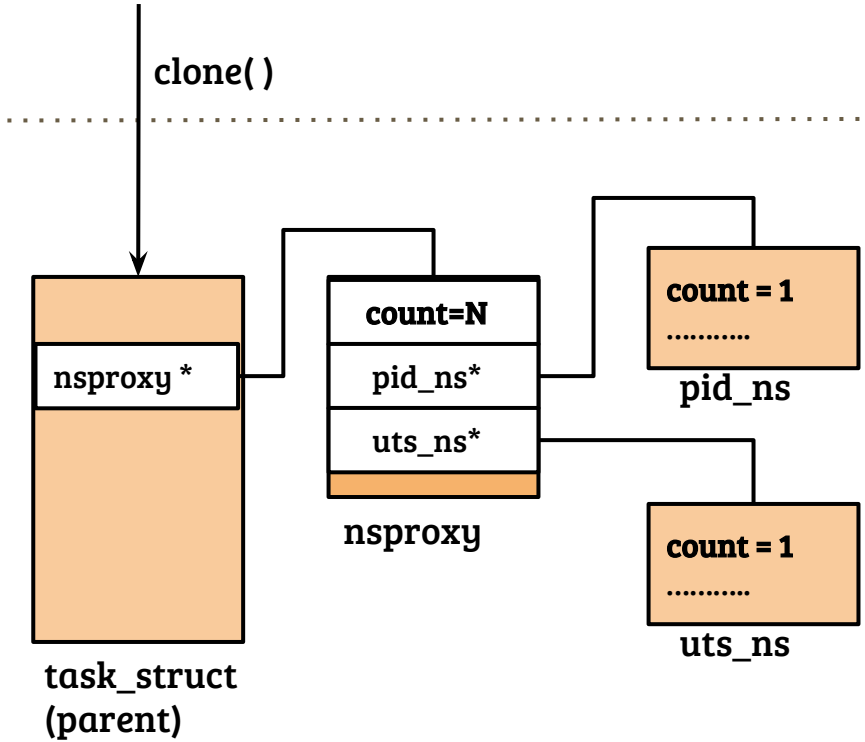
```
struct task_struct {
    struct thread_info thread_info;
    void *stack;
    ……..
    struct nsproxy *nsproxy;
    ……..
};
```

```
struct mnt_namespace {
    atomic_t count;
    struct ns_common ns;
    struct mount *root;
    ……..
    unsigned int mounts;
};
```

```
struct nsproxy {
    atomic_t count;
    struct uts_namespace *uts;
    struct pid_namespace *pid_ns;
    struct mnt_namespace *mnt_ns;
    ……..
};
```

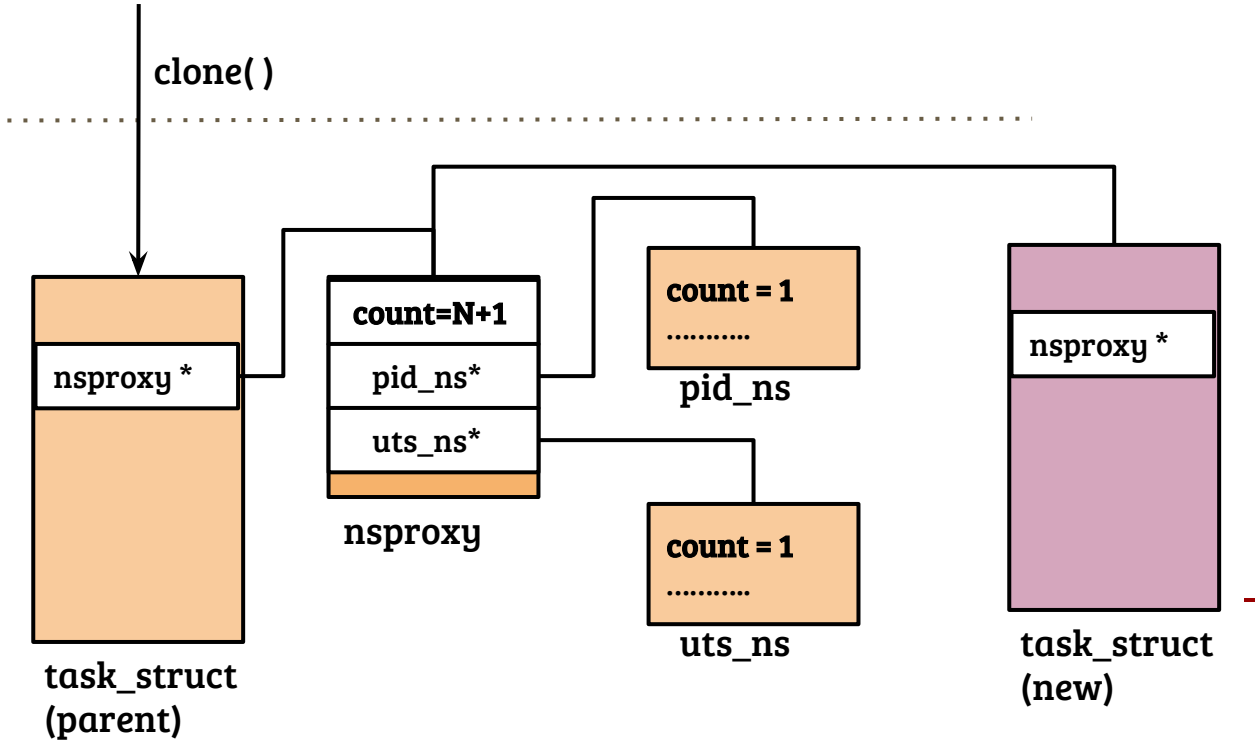
```
struct uts_namespace {
    struct kref kref;
    struct new_utsname name;
    ……..
    struct ns_common ns;
};
```

Namespaces propagation



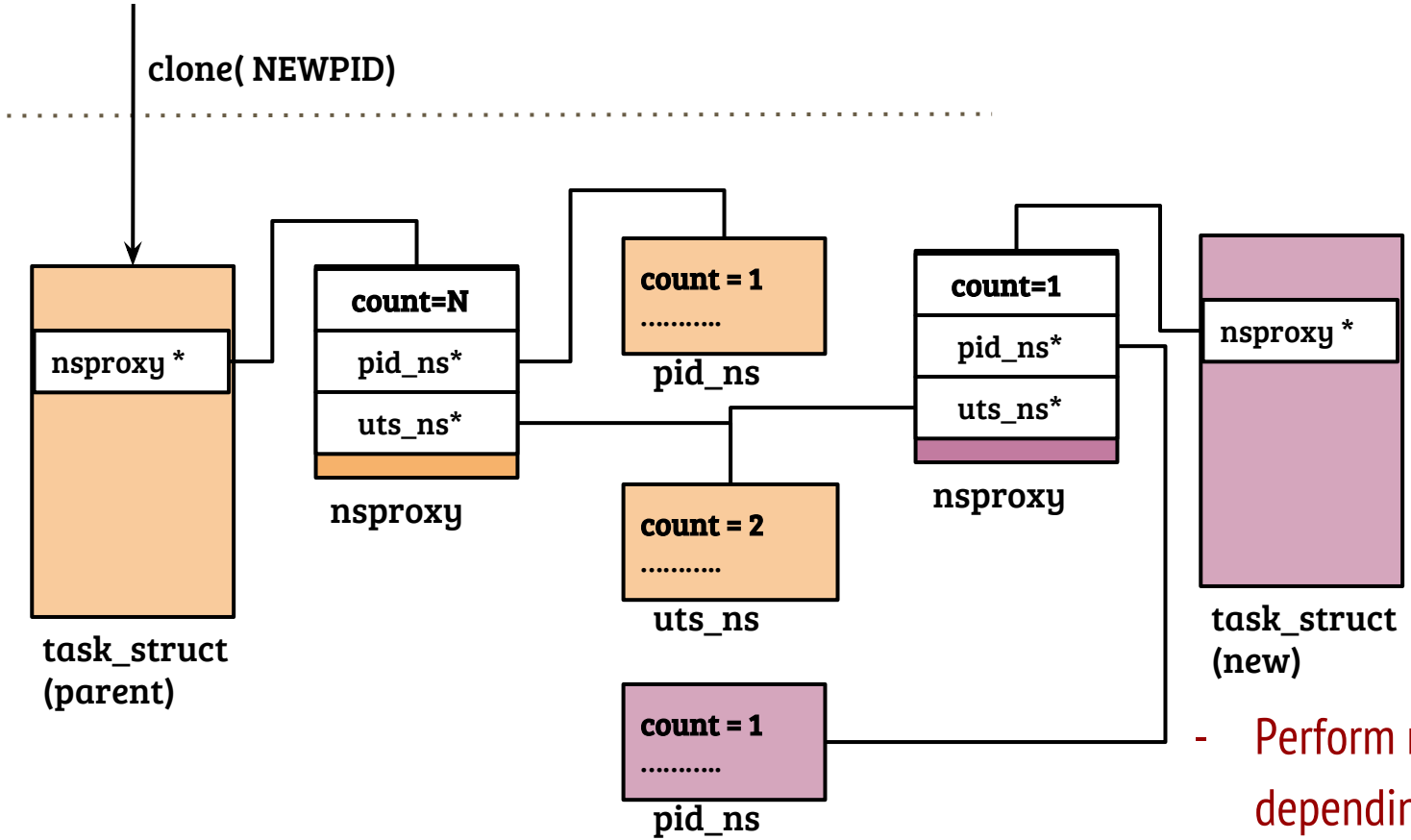
- Namespace of “original” init refers to global namespaces
- Clone with namespace flags are not very common ⇒ Avoid copy
- Refcount is a common mechanism in kernel to share a data structures w/o allocating a new copy
  - **get** ⇒ atomic_inc(count)
  - **put** ⇒ atomic_dec(count)
  - Free on **put** if count is zero

Namespaces propagation (w/o new NS flags)



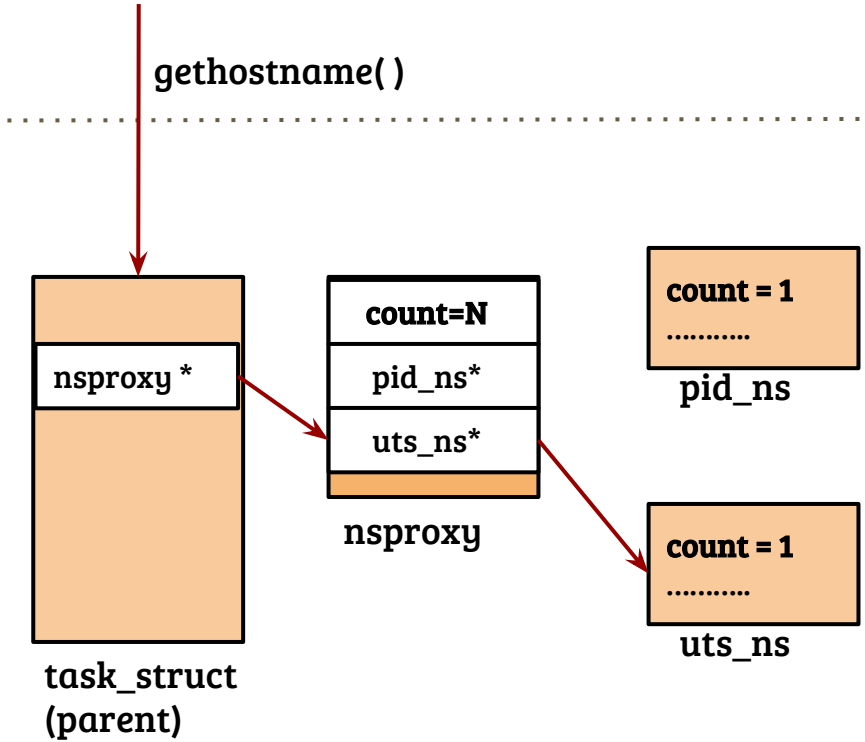
- Clone without new NS flags propagates all the namespaces efficiently

Namespaces propagation (with new NS flags)



- Perform minimal copy depending on clone flags

Namespace operation



- current⟶ nsproxy ⟶ uts_ns ⟶ hostname
- Copy the namespace to the user buffer argument
- **sethostname( )** implementation is similar with additional checks

A little bit of kernel code flow (4.19.13)

CLONE kernel/fork.c : _do_fork( ) ⇒ kernel/fork.c: copy_process( ) ⇒ kernel/nsproxy.c: copy_namespaces ⇒ kernel/nsproxy.c : create_new_namespaces( ) (conditional) ⇒ clone_*_ns()

UNSHARE kernel/fork.c : ksys_unshare( ) ⇒ kernel/nsproxy.c : unshare_nsproxy_namespaces( )

SETNS kernel/nsporxy.c: syscall_setns ⇒ create_new_namespaces( ) ⇒ ns⟶ ops ⟶ install (example install implementation in kernel/utsname.c : utsns_install)

Cgroups in kernel: data structures

```
struct task_struct {
    struct thread_info thread_info;
    void *stack;
    ……..
    struct css_set *cgroups;
    ……..
};
```

```
struct cgroup_subsys_state {
        struct cgroup *cgroup;
        struct cgroup_subsys *ss;
        ……
}
```

```
struct css_set {
  struct cgroup_subsys_state
*subsys[CGROUP_SUBSYS_COUNT];
   refcount_t refcount;
   ……..
   };
```

```
struct mem_cgroup {
    struct cgroup_subsys_state css;
    struct page_counter *mem;
    ……..
};
```

Cgroups in kernel: memory cgroup example

- While allocating a physical page for a process, update the cgroup counters
  - Example: while handling page fault ⇒ mem_cgroup_try_charge( )
  - Get a handle of the mem_cgroup
  - Update counters, trigger eviction if required ⇒ try_to_free_mem_cgroup_pages( )
- Actual logic is more complex, refer include/linux/memcontrol.h and mm/memcontrol.c