

Generic Process Containers

Paul Menage, Google
menage@google.com

OLS 2007



Generic Process Containers

- Motivation
- Requirements
- Design Overview
- APIs and Implementation
- Examples
- Future Work



Motivation

- Linux provides per-process, per-user controls
- Many projects need controls on definable groups of processes
- We need a “VFS” for process tracking
- “Containers” - what’s in a name?



Requirements

- Extensibility - easy to add *subsystems*
- Definable containment
- Minimal overhead
- Flexible userspace API
- Multiple partitions
 - Each subsystem bound to at most one partition



Design Overview

- *Hierarchy* - a partition across processes
- *Container* - a group defined in a hierarchy
- A process is in one container in each hierarchy
- A *subsystem* may be attached to a hierarchy
- *Subsystem-state* provides per-group state



Subsystem API

- Each subsystem provides callbacks
- Required:
 - `create()`, `destroy()`
- Optional:
 - `can_attach()`, `attach()`
 - `fork()`, `exit()`
 - `populate()`
- Flexible userspace API
 - Long term, some standardization needed



Implementation

- struct containerfs_root
 - One instance per hierarchy
 - super_block->s_fsinfo
 - One or more bound subsystems, as mount options
 - Mounted on zero or more VFS locations
- struct container
 - One per container grouping in an active hierarchy
 - dentry->d_fsdata
 - Holds pointers to per-subsystem states

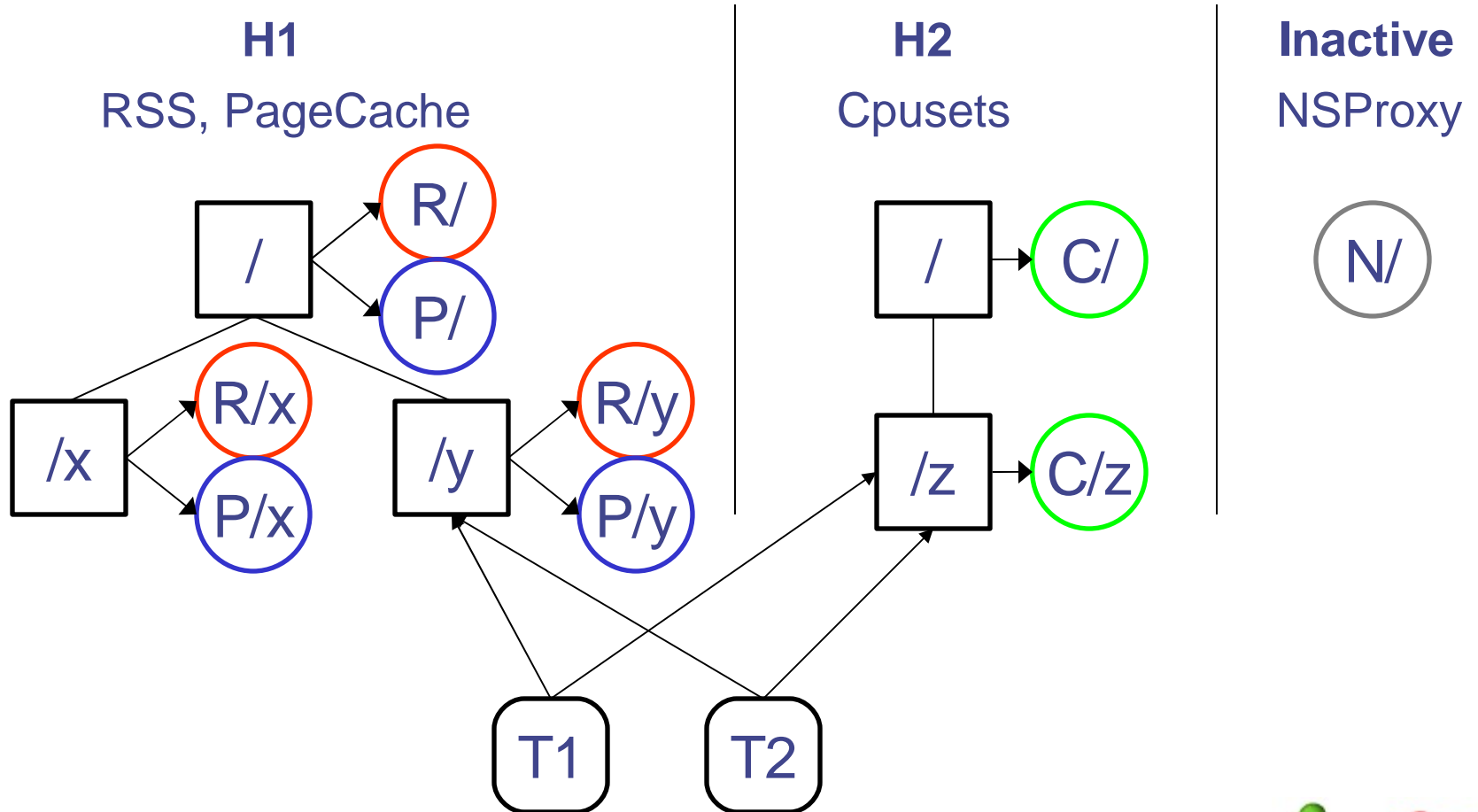


Implementation

- `struct container_subsys`
 - One per registered container subsystem
 - Holds callback table
 - Registered statically at compile time
- `struct container_subsys_state`
 - Embedded in subsystem-specific state object
 - One per container, per bound subsystem
 - Holds generic housekeeping data



Implementation

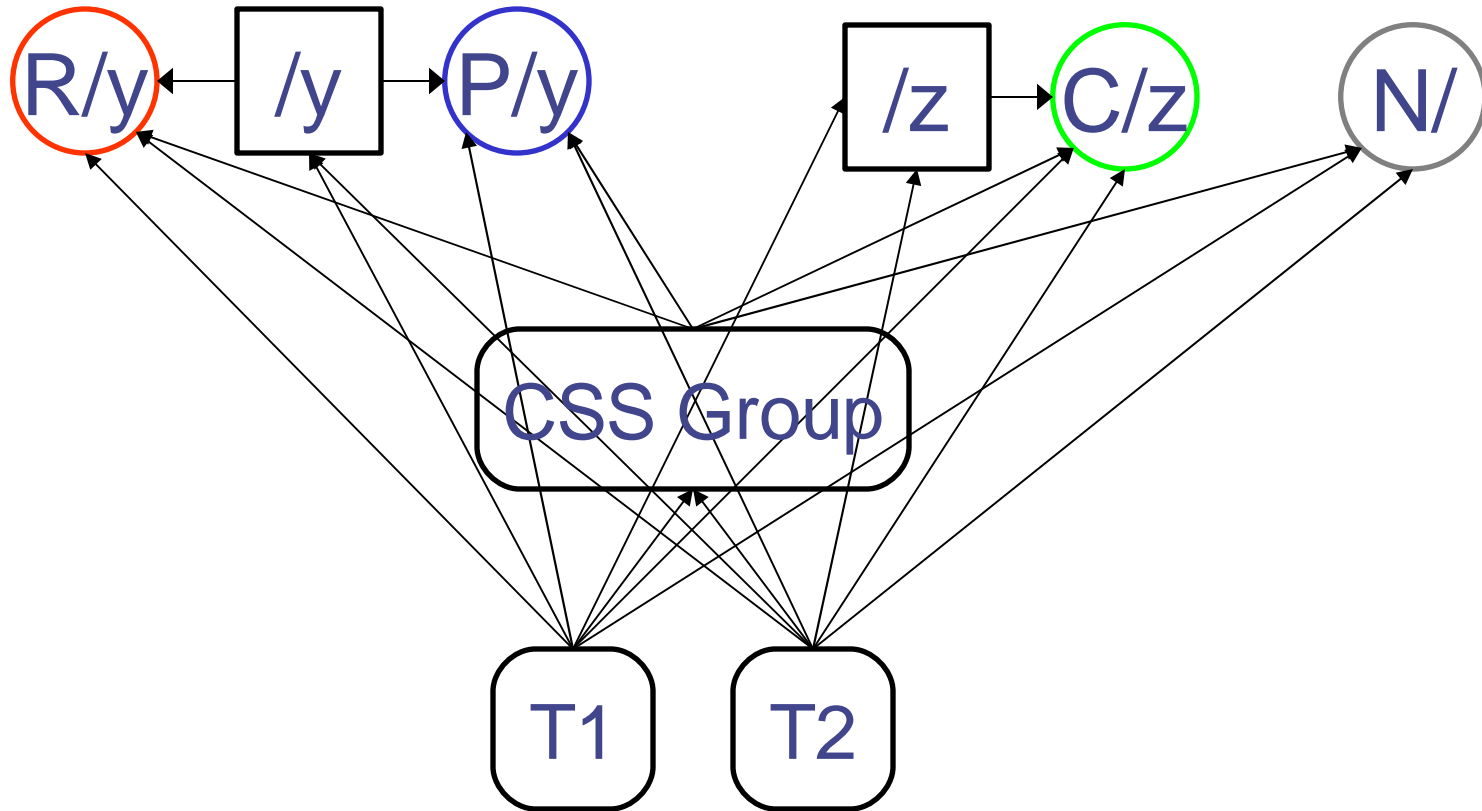


Implementation

- `task_struct` points to `container_subsys_state`
 - Directly accessible for fast path operations
 - container object indirectly accessible for slow path



Implementation



Implementation

- `css_group` (container subsystem state group)
 - Holds pointers to `container_subsys_state` objects
 - Shared by tasks with same container memberships
 - Reduced space overhead in `task_struct`
 - Reduced ref-counting at `fork()/exit()`



Performance

- Goal: minimal overhead in fast-path operations
 - clone()/exit() housekeeping
 - Benchmarked with a NOP pthread_create/pthread_join()
 - indirection accessing subsystem state
 - Benchmarked by faulting on an anonymous mapping
- Benchmarks showed no overhead for cpusets
 - Exception: css_group task linked list
 - Can be optimized via lazy list construction



Example Subsystems

- Cpusets
- CPU accounting
- Resource Counters - container library
- RSS and PageCache controllers
- Namespace subsystem
- CPU scheduler interface
- Task freezer
- CKRM / BeanCounters libraries



Future Work

- More subsystems
- Optimizations
- API conventions
- Pushing towards mainline tree

