

LCA 2019

What's new in control groups (cgroups) v2

Michael Kerrisk, man7.org © 2019
mtk@man7.org

linux.conf.au 23 January 2019
Christchurch, New Zealand

Outline

1	Introduction	3
2	Problems with cgroups v1; rationale for v2	7
3	Cgroups v2 controllers	19
4	Enabling and disabling controllers	24
5	Organizing cgroups and processes	33
6	Optional topic: release notification (cgroup.events file)	38
7	Optional topic: delegation	41
8	Optional topic: overview of thread mode	48

Outline

1	Introduction	3
2	Problems with cgroups v1; rationale for v2	7
3	Cgroups v2 controllers	19
4	Enabling and disabling controllers	24
5	Organizing cgroups and processes	33
6	Optional topic: release notification (cgroup.events file)	38
7	Optional topic: delegation	41
8	Optional topic: overview of thread mode	48

Who am I?

- Contributor to Linux *man-pages* project since 2000
 - Maintainer since 2004
 - Maintainer email: `mtk.manpages@gmail.com`
 - Project provides ≈ 1050 manual pages, primarily documenting system calls and C library functions
 - <https://www.kernel.org/doc/man-pages/>
- Author of a book on the Linux programming interface
 - <http://man7.org/tlpi/>
- Trainer/writer/engineer
 - Lots of courses at <http://man7.org/training/>
- Email: `mtk@man7.org`
Twitter: `@mkerrisk`

Assumptions

- You have a basic understanding of the purpose of cgroups (control groups)
- You have some familiarity with cgroups v1

Outline

- Topics:
 - Problems with cgroups v1 / rationale for cgroups v2
 - Brief overview of controllers in v2
 - V2 differences:
 - Enabling/disabling controllers
 - Organizing processes within v2 hierarchy
- Other topics, as time permits:
 - Release notification
 - Delegation
 - Thread mode
- Questions: at the end (if we have time)

Outline

1	Introduction	3
2	Problems with cgroups v1; rationale for v2	7
3	Cgroups v2 controllers	19
4	Enabling and disabling controllers	24
5	Organizing cgroups and processes	33
6	Optional topic: release notification (cgroup.events file)	38
7	Optional topic: delegation	41
8	Optional topic: overview of thread mode	48

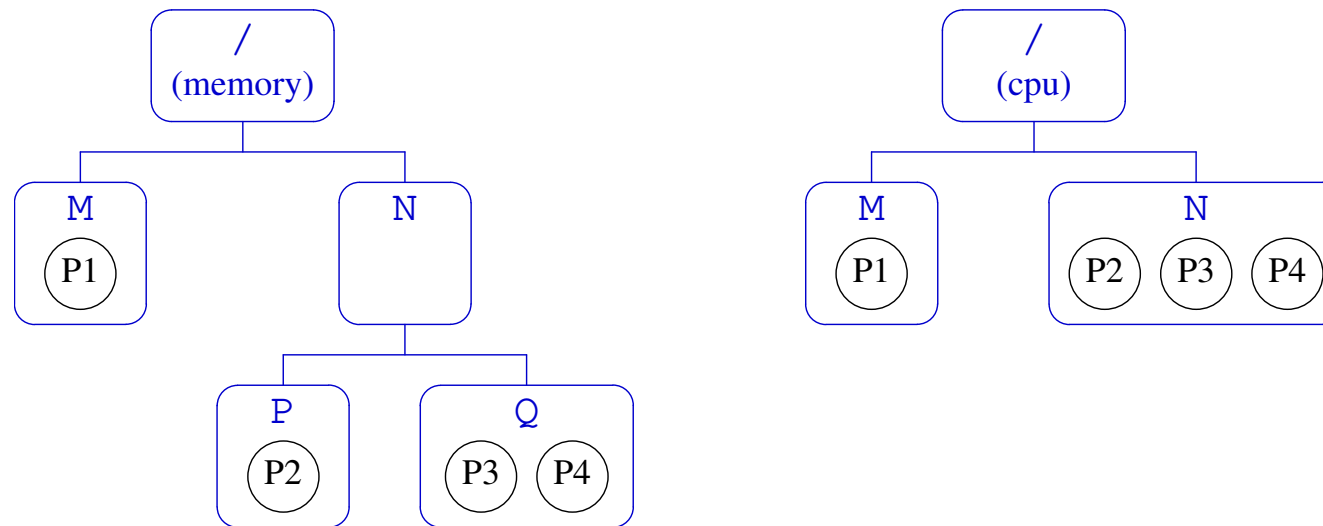
Cgroups version 2

- Designed to address perceived problems with cgroups v1
 - Problems sprang from lack of any coordinated design in cgroups v1 controllers
- Officially released in Linux 4.5 (March 2016)
 - After lengthy development phase that commenced in 2012
- Can use both cgroups v1 and cgroups v2 on same system
 - **But** can't mount same controller in both filesystems
- Further information on cgroups v2:
 - `Documentation/cgroup-v2.txt` kernel source file
 - `cgroups(7)` manual page

Problems with cgroups v1: multiple hierarchies

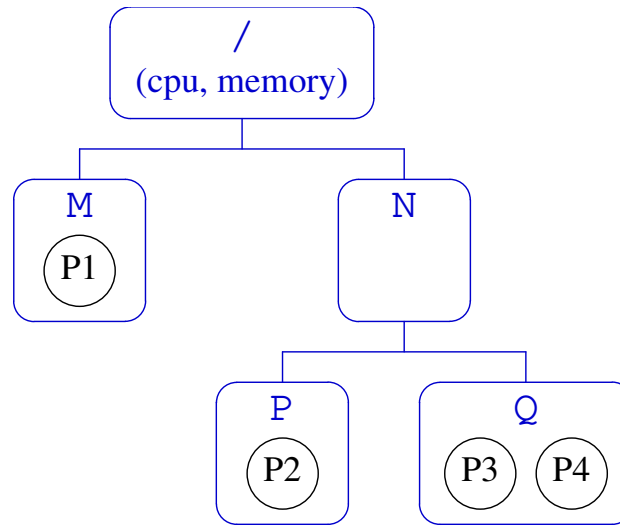
- **V1 hierarchy scheme** was supposed to allow great flexibility
 - V1: arbitrary number of hierarchies; one or more controllers can be mounted against each hierarchy
- But, that flexibility was **less useful than originally envisaged**
- Let's consider pros and cons of two approaches:
 - Separate hierarchy per controller
 - Attaching multiple controllers to same hierarchy

Attaching v1 controllers to separate hierarchies



- 😊 Attaching controllers to separate hierarchies means they can **manage processes at different granularities**
 - memory can finely control memory allocation for P2 vs. P3 + P4
 - cpu allows P2 + P3 + P4 to share a CPU allocation ($\frac{1}{3}$ each)
- 😞 But when **moving process across cgroups** (e.g., moving P2 to cgroup M), operation **must be repeated in each hierarchy**
 - Cumbersome, slow, and nonatomic

Attaching multiple v1 controllers to the same hierarchy



- 😊 Placing multiple controllers on same hierarchy removes need to replicate move operations in multiple hierarchies
- 😞 But, **controllers must manage to same level of granularity**
 - E.g., $P2 + P3 + P4$ can no longer share a CPU allocation
 - Must make specific allocation decisions for $P2$ vs $P3 + P4$
 - (Note: establishing CPU limit in N isn't sufficient: its allocation will be split **equally** between P and Q)

Problems with cgroups v1: multiple hierarchies

Other problems with the v1 hierarchy scheme:

- ☹ Utility controllers (e.g., freezer) that might be useful in all hierarchies could be used in only one
 - E.g., to freeze all processes in a cpu cgroup, there must be a freezer cgroup with same membership
 - And same is true if we want to freeze a memory cgroup, etc.
- Argues in favor of attaching all controllers to same hierarchy or maintaining parallel hierarchies that are highly similar

Problems with cgroups v1: multiple hierarchies

- In most use cases, **completely orthogonal (i.e., nonparallel) hierarchies were not needed**
- More common requirement: have **different levels of granularity per controller**
 - E.g., control memory only to a certain level in tree, but provide finer-grained control of CPU at deeper levels
- ⇒ Applications commonly put **most controllers on separate, but highly similar, hierarchies**
 - Topology of trees differed in cases where different granularity of control was needed

Problems with cgroups v1: multiple hierarchies

- ⇒ **v2 uses single hierarchy for all controllers**
 - Establish common domain for all resource types, so controllers can cooperate
 - And there is a mechanism to allow per-controller granularity in the hierarchy

Problems with cgroups v1: thread granularity

Allowing **thread granularity** for cgroup membership proved problematic

- Main problem: it doesn't make sense for some controllers
 - E.g., memory controller (threads share memory...)
- ⇒ **v2 allows only process-granularity** membership
 - But starting with Linux 4.14, there is a limited form of thread granularity for some controllers...

Problems with cgroups v1: cgroups vs tasks

- Allowing a cgroup to contain both tasks and child cgroups was problematic in some cases
 - Two different types of entities—*tasks* and *groups* of tasks—compete for distribution of same resources
 - Different controllers interpreted this in differing ways...
 - which caused difficulties if trying to combine multiple controllers on same hierarchy / closely parallel hierarchies
 - ⇒ **In v2, only leaf cgroups can contain processes**
 - (The story is more subtle; we'll revisit)

Problems with cgroups v1: inconsistency

- **Inconsistencies** between controllers (“design followed implementation”)
 - With some controllers, new cgroups inherit parent’s attributes; in others, they get defaults
 - Some controllers have controller-specific interfaces in root cgroup; others don’t
 - Inconsistent use of values in cgroup files (e.g., “maximum” represented as “-1” vs “max”)
- v2: **consistent names and values** for interface files, **consistent inheritance rules** for all controllers
 - With some clearly documented guidelines!

Problems with cgroups v1: cgroup release notification

- Release notification == ability to get notified when last process leaves a cgroup
- V1 cgroup release notification mechanism has problems:
 - A process is fired up on each release \Rightarrow expensive!
 - Can't delegate release handling to process inside a container
 - \Rightarrow **v2 has a lightweight solution** that supports delegation

Outline

1	Introduction	3
2	Problems with cgroups v1; rationale for v2	7
3	Cgroups v2 controllers	19
4	Enabling and disabling controllers	24
5	Organizing cgroups and processes	33
6	Optional topic: release notification (cgroup.events file)	38
7	Optional topic: delegation	41
8	Optional topic: overview of thread mode	48

Cgroups v2 controllers

- By now, v2 is pretty much ready for prime time
 - There are equivalents of nearly all v1 controllers
 - In some cases, v2 controllers are nearly identical to v1
 - Typically because v1 controller was added during 3½-year development phase of v2
 - Other v2 controllers are significant redesigns
 - Based on lessons learned from v1
- `Documentation/cgroup-v2.txt` documents v2 controllers

Controllers available in cgroups v2

- `memory`: control distribution of memory
 - Successor to v1 memory controller
- `io`: regulate distribution of I/O resources
 - Successor to v1 blkio controller
- `pids`: control number of processes
 - Exactly the same as v1 controller
- `perf_event`: per-cgroup *perf* monitoring (since Linux 4.11)
 - Same as v1 controller (added in same kernel version)
- `rdma`: distribution and accounting of RDMA resources (since Linux 4.11)
 - Same as v1 controller (added in same kernel version)

Controllers available in cgroups v2

- `cpu`: successor to v1 `cpu` and `cpuacct` controllers (since Linux 4.15)
 - **Lack of this controller was a roadblock** for v2 adoption
- `devices`: control access to devices (since Linux 4.15)
 - Successor to v1 `devices` controller
 - No interfaces files; instead control is done by attaching eBPF (`BPF_CGROUP_DEVICE`) program to cgroup
 - Each attempt to access device is gated by decision that eBPF program returns to kernel
- `cpuset`: successor to v1 `cpuset` controller (since Linux 5.0)
- No direct equivalent of `net_cls` + `net_prio`
 - Instead, support was added in *iptables* to allow BPF filters that hook on cgroup v2 pathnames to allow control of NW traffic on a per-cgroup basis
 - Since Linux 4.5(?)

Controllers not (so far) available in cgroups v2

- As at Linux 5.0, v2 currently lacks equivalents of:
 - freezer (“soon”)
 - hugetlb (was problematic; may simply be dropped, as there are preferable alternatives)

Outline

1	Introduction	3
2	Problems with cgroups v1; rationale for v2	7
3	Cgroups v2 controllers	19
4	Enabling and disabling controllers	24
5	Organizing cgroups and processes	33
6	Optional topic: release notification (cgroup.events file)	38
7	Optional topic: delegation	41
8	Optional topic: overview of thread mode	48

Mounting the cgroups v2 filesystem

- To use cgroups v2, we mount new filesystem type:

```
# mount -t cgroup2 none /path/to/mount
```

- Recent *systemd* automatically creates such a mount point, at `/sys/fs/cgroup/unified`
- All **v2 controllers are automatically available** under single hierarchy
 - No need to explicitly bind controllers to mount point
 - I.e., we don't specify `-o controller` mount option

The `cgroup.controllers` file

- Each v2 cgroup has a (read-only) `cgroup.controllers` file, which lists **available controllers** this cgroup can enable
- But, if we look in cgroups v2 root directory, we might find `cgroup.controllers` is empty:

```
# cd /sys/fs/cgroup/unified
# cat cgroup.controllers
# wc -l cgroup.controllers
0 cgroup.controllers
```

- ... because v2 controller is available only if not bound in v1 hierarchy

```
# cat /proc/mounts | grep pids
cgroup /sys/fs/cgroup/pids cgroup rw,....,pids 0 0
```

- That's why we don't see `pids` in `cgroup.controllers`

Ensuring that a controller is available in cgroups v2

- May need to unmount controller in v1 hierarchy to have it available in v2 hierarchy:

```
# umount /sys/fs/cgroup/pids
# cat /sys/fs/cgroup/unified/cgroup.controllers
pids
```

- But cgroup v1 FS can be successfully unmounted only if:
 - All processes are in root cgroup
 - There are no child cgroups
 - No process has open FDs or CWD on filesystem
 - `cgroups/remove_cgroup_hier.sh` provides example of performing following steps for a v1 hierarchy:
 - Moving all processes to root cgroup
 - Removing all child cgroups (from bottom up)



Ensuring that a controller is available in cgroups v2

- Alternatively, (since Linux 4.6) use kernel boot parameter, `cgroup_no_v1`:
 - `cgroup_no_v1=all` \Rightarrow disable all v1 controllers
 - `cgroup_no_v1=controller,...` \Rightarrow disable selected controllers
(*systemd* falls back ok if no v1 controllers are available)

Enabling and disabling controllers

- Controllers are enabled/disabled by writing some subset of available controllers to `cgroup.subtree_control`

```
# echo "+pids -memory" > cgroup.subtree_control
```

- + \Rightarrow enable controller, - \Rightarrow disable controller
- Enabling a controller in `cgroup.subtree_control`:
 - Allows resource to be **controlled in child cgroups**
 - **Creates controller-specific attribute files in each child directory**
-   Attribute files in child cgroups are **used by process managing parent cgroup** to manage resource allocation across child cgroups
 - Different from v1...

Example: enabling a controller

- In the cgroup root directory, list available controllers:

```
# cat cgroup.controllers
cpu io memory pids
```

- Create a child cgroup; see what files are in subdirectory:

```
# mkdir grp1
# ls grp1
cgroup.controllers  cgroup.events  cgroup.procs
cgroup.subtree_control
```

- Enable pids controller for child cgroups; new control files have been created in child cgroup:

```
# echo '+pids' > cgroup.subtree_control
# ls grp1
cgroup.controllers  cgroup.subtree_control  pids.max
cgroup.events       pids.current
cgroup.procs        pids.events
```

Example: enabling a controller

- In grp1 cgroup, only available controller is pids:

```
# cat grp1/cgroup.controllers
pids
```

- In child of grp1, we can enable pids controller:

```
# mkdir grp1/sub
# echo '+pids' > grp1/cgroup.subtree_control
# cat grp1/cgroup.subtree_control
pids
```

- But io controller is not available:

```
# echo '+io' > grp1/cgroup.subtree_control
sh: echo: write error: No such file or directory
```

- ENOENT error because “entry we are trying to add to subtree_control does not exist in controllers”

Top-down constraints

- Child cgroups are always subject to any resource constraints established by controllers in ancestor cgroups
 - \Rightarrow Descendant cgroups can't relax constraints imposed by ancestor cgroups
- If a controller is disabled in a cgroup (i.e., not written to `cgroup.subtree_control` in parent cgroup), it cannot be enabled in any descendants of the cgroup

Outline

1	Introduction	3
2	Problems with cgroups v1; rationale for v2	7
3	Cgroups v2 controllers	19
4	Enabling and disabling controllers	24
5	Organizing cgroups and processes	33
6	Optional topic: release notification (cgroup.events file)	38
7	Optional topic: delegation	41
8	Optional topic: overview of thread mode	48

Organizing cgroups and processes

Broadly similar to cgroups v1:

- Hierarchy organized as set of subdirectories
- All processes initially in root cgroup
- Move process into group by writing PID into `cgroup.procs`
- Read `cgroup.procs` to discover process membership
- Child of `fork()` inherits parent's cgroup membership
- Cgroup directory with no (non-zombie) process members or child cgroups can be removed

Organizing cgroups and processes

Differences between v1 and v2:

- Root cgroup does not contain controller interface files
 - Control is not exercised on processes in root cgroup
- Cgroup can't both control cgroup children and have member processes
 - \Rightarrow Place member processes in leaf nodes
- In initial implementation, cgroups v2 supported only process-level granularity
 - From Linux 4.14, a limited form of thread-granularity cgroup membership is restored for certain controllers
 - So-called “thread mode”

“Only leaf nodes can have member processes”

- Earlier statement: “a cgroup can’t have both child cgroups and member processes”
- Let’s refine that...
- A cgroup can’t both:
 - distribute a resource to child cgroups (i.e., enable controllers in `cgroup.subtree_control`), **and**
 - have child processes

“Only leaf nodes can have member processes”

- Revised statement: “A cgroup can’t both distribute resources and have member processes”
- Conversely (1):
 - A cgroup **can** have member processes and child cgroups...
 - **iff** it does not enable controllers for child cgroups
- Conversely (2):
 - If cgroup has child cgroups and processes, the processes must be moved elsewhere before enabling controllers
 - E.g., processes could be moved to child cgroups
- ⚠ This rule changes for certain controllers in Linux 4.14
 - (The so-called “threaded” controllers)

Outline

1	Introduction	3
2	Problems with cgroups v1; rationale for v2	7
3	Cgroups v2 controllers	19
4	Enabling and disabling controllers	24
5	Organizing cgroups and processes	33
6	Optional topic: release notification (cgroup.events file)	38
7	Optional topic: delegation	41
8	Optional topic: overview of thread mode	48

Cgroup (un)populated notification

- Cgroups v1: firing up a process is an expensive way of get notification of an empty cgroup!
 - Also: release agent setting is per hierarchy
 - (Can't have different release agents for different subtrees of a hierarchy)
- Cgroups v2: dispenses with v1's `release_agent` and `notify_on_release` files
- Instead, each (non-root) cgroup has a file, `cgroup.events`, with a populated field:

```
# cat grp1/cgroup.events
populated 1
```

- 1 == subhierarchy contains live processes
 - I.e., live process in cgroup, or in any descendant cgroup
- 0 == no live processes in subhierarchy

Cgroup (un)populated notification

- Can monitor `cgroup.events` file, to get notification of transition between populated and unpopulated states
 - *inotify*: transitions generate `IN_MODIFY` events
 - *poll()/epoll*: transitions generate `POLLPRI/EPOLLPRI` events
- One process can monitor multiple `cgroup.events` files
 - Much cheaper notification!
 - **Notification can be delegated** per container
 - I.e., one process can monitor all `cgroup.events` files in a subhierarchy

Outline

1	Introduction	3
2	Problems with cgroups v1; rationale for v2	7
3	Cgroups v2 controllers	19
4	Enabling and disabling controllers	24
5	Organizing cgroups and processes	33
6	Optional topic: release notification (cgroup.events file)	38
7	Optional topic: delegation	41
8	Optional topic: overview of thread mode	48

Delegation

- Delegation == passing management of some subtree of hierarchy to another (less privileged) user
 - I.e., some other user who will manage resource control in the subhierarchy of processes
 - Useful for containers run by non-*root* users
- Terminology:
 - **Delegater**: privileged user who owns a parent cgroup
 - **Delegatee**: less privileged user who is assigned management of a subhierarchy under parent cgroup

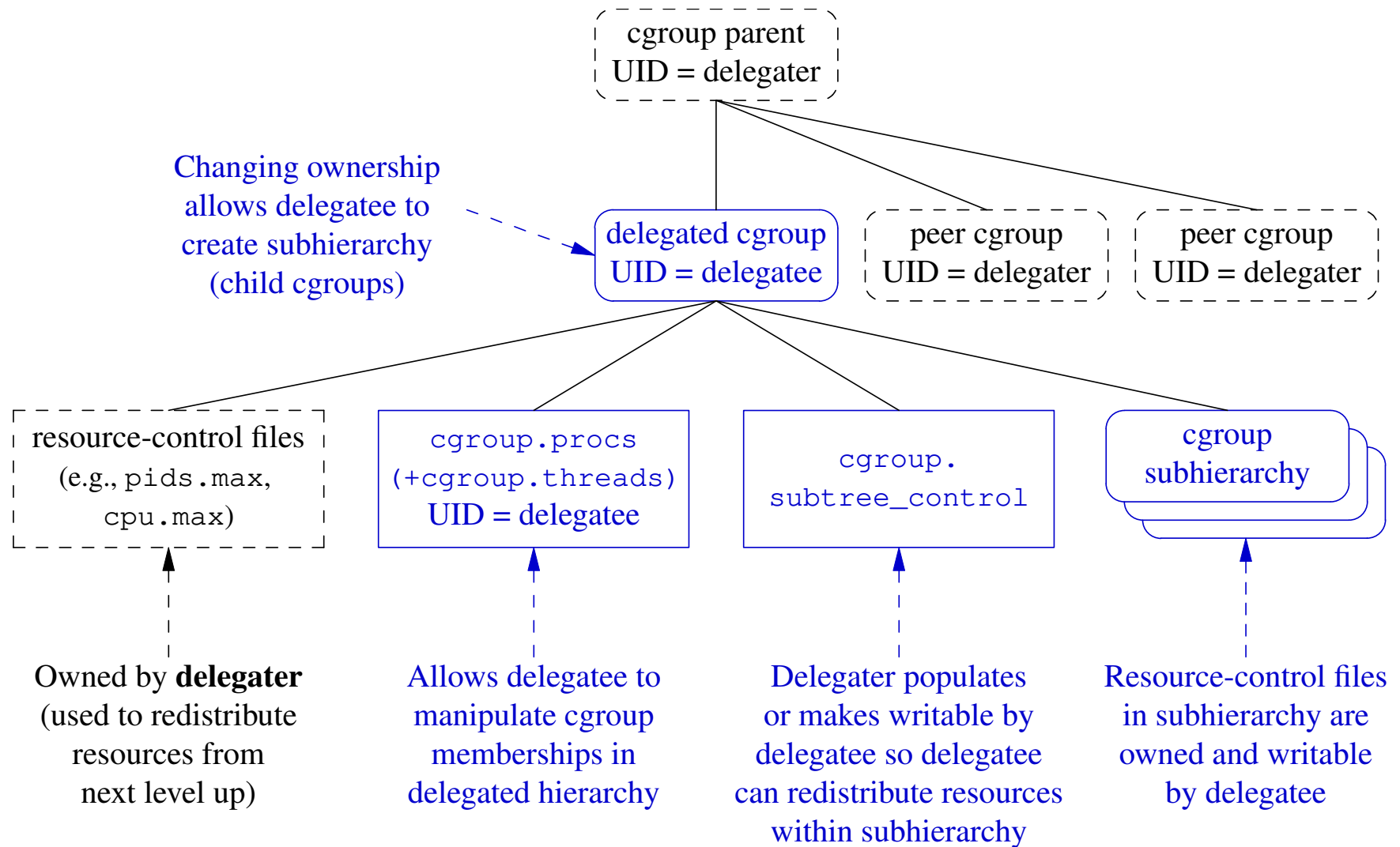
Delegation set-up

- To set up delegation, delegater grants delegatee write access to certain files
 - Normally done by changing ownership to UID of delegatee
- In addition to directory at root of delegated subtree, ownership of following files inside that directory is changed:
 - `cgroups.procs`
 - `cgroup.subtree_control`
 - So that delegatee can control resources in child cgroups it creates
 - `cgroup.threads`, if delegating a threaded subtree
 - + any other files listed in `/sys/kernel/cgroup/delegate`

Delegation set-up

- ⚠ Delegater **should not** make resource-control interface files writable by delegatee
 - Those files are used by **parent** (delegater) to control resource allocation in the child (delegatee)
 - ⇒ Delegatee should not have permission to change them

Delegation set-up



Post-delegation operation

- After delegation, delegatee can:
 - Create subhierarchy under delegated cgroup
 - Move process between cgroups inside subhierarchy
 - But, **delegation containment rules** mean delegatee can't move process into/out of subhierarchy (see *cgroups(7)*)
 - Control distribution of resources in subhierarchy
 - If controller is present in `cgroup.subtree_control`

Delegation in cgroups v1

- Delegation concept exists in cgroups v1
- But, was an accidental by-product of the filesystem-based implementation
- Delegation in v1 doesn't have proper containment rules
 - Reportedly, there are also other security issues

Outline

1	Introduction	3
2	Problems with cgroups v1; rationale for v2	7
3	Cgroups v2 controllers	19
4	Enabling and disabling controllers	24
5	Organizing cgroups and processes	33
6	Optional topic: release notification (cgroup.events file)	38
7	Optional topic: delegation	41
8	Optional topic: overview of thread mode	48

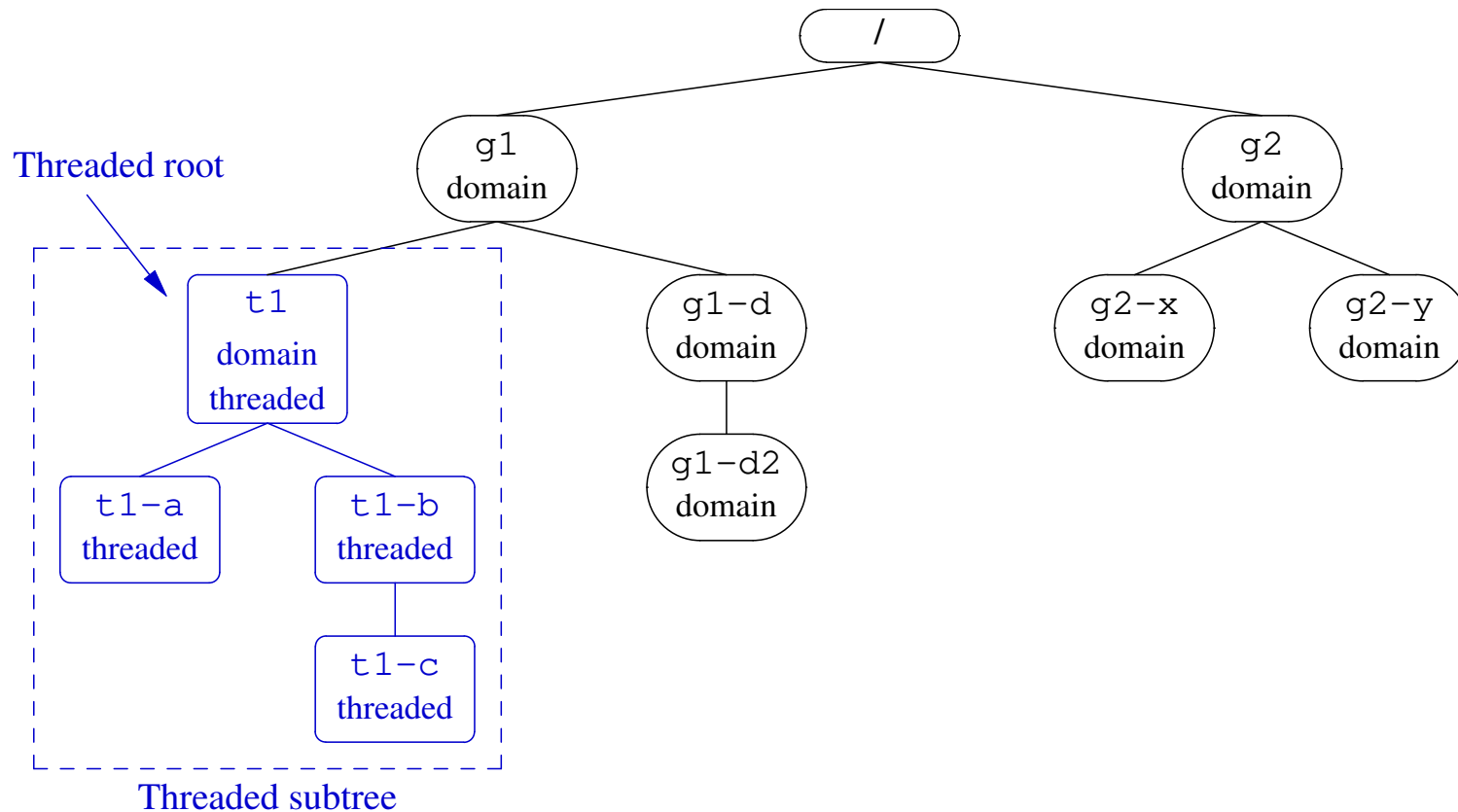
Background

- Original design goal in v2: all threads in multithreaded (MT) process are always in same cgroup
- By contrast, v1 permitted threads to be split across cgroups
 - But, this made no sense for some controllers (e.g., memory)
- Despite the initial v2 design decision, there were use cases for thread-level control with cpu controller
- Result was a stand-off for a long period:
 - Cgroups v2 developers: “control is only at process level”
 - Kernel scheduler maintainers: “we won’t merge a v2 cpu controller that doesn’t allow thread-granularity control”
- Solution: **thread mode**, added in Linux 4.14
 - Allows thread-level granularity for certain controllers

“domain” versus “threaded” cgroups

- Cgroups in v2 hierarchy are initially all in “domain” mode:
 - All threads in MT process must be in same cgroup
 - This is the original cgroup v2 default
- Selected **subtrees** of hierarchy can be switched to “threaded” mode
 - All members of subtree must be “threaded” cgroups
 - Threads of MT processes can be in different cgroups under a “threaded” subtree
 - Restriction: all threads of a MT process must be inside **same** “threaded” subtree
- There can be multiple “threaded” subtrees, each containing multiple processes
- Thus, v2 now has thread granularity, but in more restricted manner than v1

Cgroup v2 thread mode



A threaded subtree within the cgroup v2 hierarchy

- Threads of MT process can be split across cgroups in threaded subtree

Threaded and domain controllers

Starting with Linux 4.14, there are two kinds of controllers...

- **Threaded** controllers: support thread-granularity control
 - `cpu`, `cpuset`, `perf_event`, `pids`
- **Domain (nonthreaded)** controllers: support only process-granularity control
 - All other controllers...

Threaded and domain controllers

- **Threaded** controllers understand threaded subtrees
 - IOW: controller-interface files for threaded controllers do appear in threaded subtrees
- To **domain** controllers, threaded subtrees are “invisible”
 - IOW: controller-interface files for domain controllers **do not** appear in threaded subtrees
 - I.e., domain controllers don't distribute resources in threaded subtree
 - From perspective of domain controllers, all threads in MT process appear to be in one cgroup—the “domain threaded” root cgroup
 - (Recall that all threads of a process must be in same threaded subtree)

New interface files for thread mode

- `cgroup.threads`: define/view thread membership of cgroup
 - Write thread ID to this file to move thread to cgroup
 - Read file to get list of threads in cgroup
- `cgroup.type`: defines type of cgroup, and contains one of:
 - `domain`: normal group providing process-granularity control
 - (I.e., the original cgroup v2 behavior)
 - `threaded`: a group that is a member of a threaded subtree
 - `domain threaded`: a domain group that serves as root of a threaded cgroup subtree
 - `domain invalid`: group in an “invalid” state
 - Can't be populated with processes and can't have controllers enabled
 - Can be converted to “threaded” group

Creating a threaded subtree

- There are two different ways of creating a threaded subtree
 - Full details are in the *cgroups(7)* manual page
- But many details and rules about how this must be done...
 - More complex than we have time to cover
 - Possible demo...
 - And use `cgroups/view_v2_cgroups.go` to inspect `cgroups`

Thanks!

Michael Kerrisk mtk@man7.org [@mkerrisk](https://twitter.com/mkerrisk)

Slides at <http://man7.org/conf/>

Source code at <http://man7.org/tlpi/code/>

Training: Linux system programming, security and isolation APIs,
and more; <http://man7.org/training/>

The Linux Programming Interface, <http://man7.org/tlpi/>

