Linux Security and Isolation APIs

Control Groups (cgroups): Introduction

Michael Kerrisk, man7.org © 2023

February 2023

mtk@man7.org

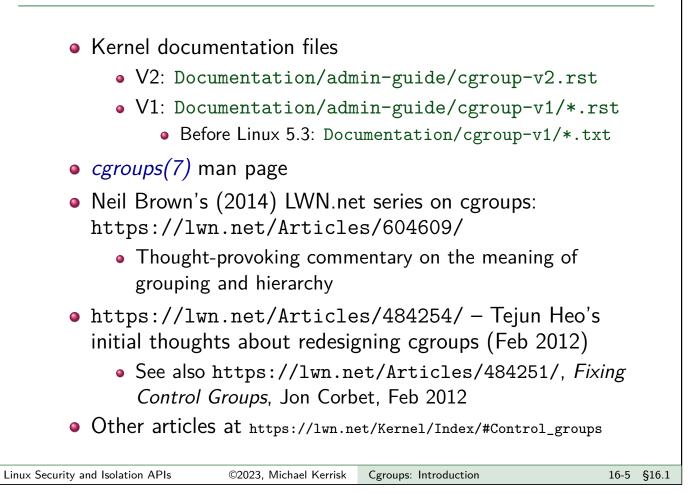
Outline	Rev: #e2bf8f005a44
16 Cgroups: Introduction	16-1
16.1 Preamble	16-3
16.2 What are control groups?	16-9
16.3 An example: the pids controller	16-16
16.4 Creating, destroying, and populating a cgroup	16-21
16.5 Enabling and disabling controllers	16-34

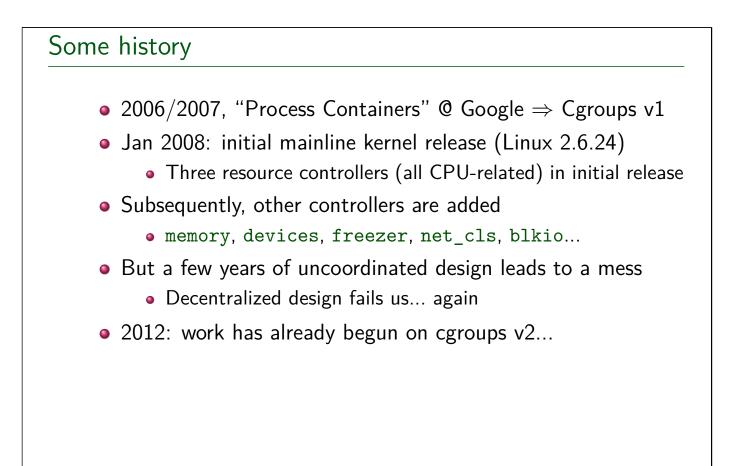
Goals

• We'll focus on:

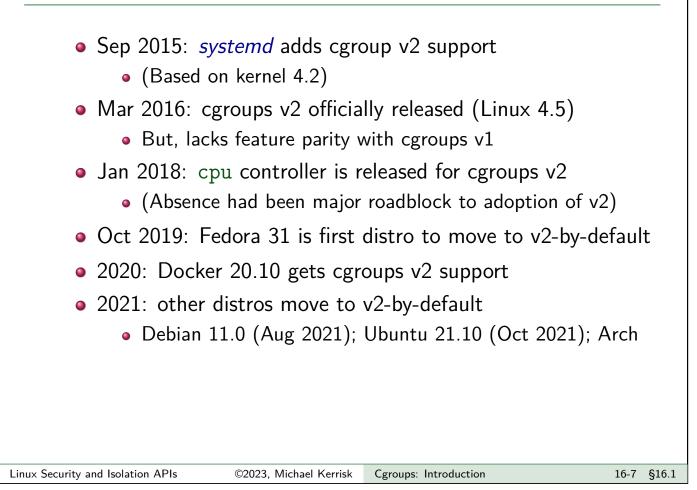
- General principles of operation; goals of cgroups
- The cgroup2 filesystem
- Interacting with cgroup2 filesystem using shell commands
- Origin of cgroups v2 (i.e., problems with cgroups v1)
- Differences between cgroups v2 and v1
- We'll look briefly at some of the controllers

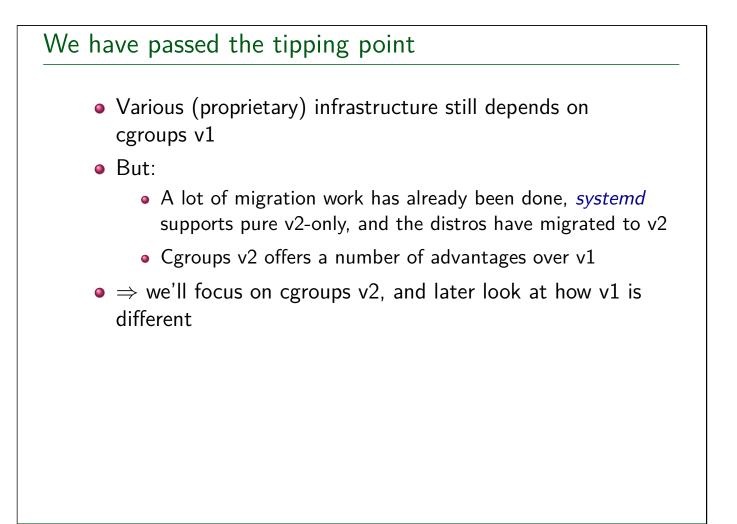
Resources





Some history



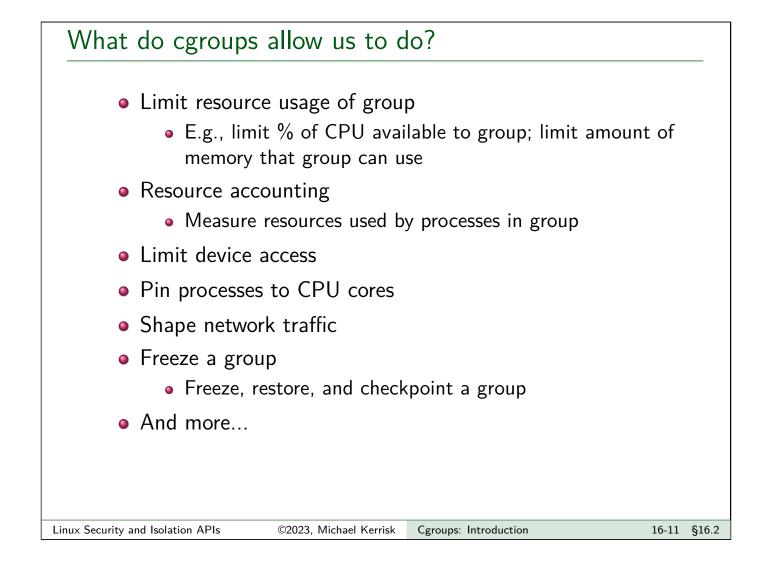


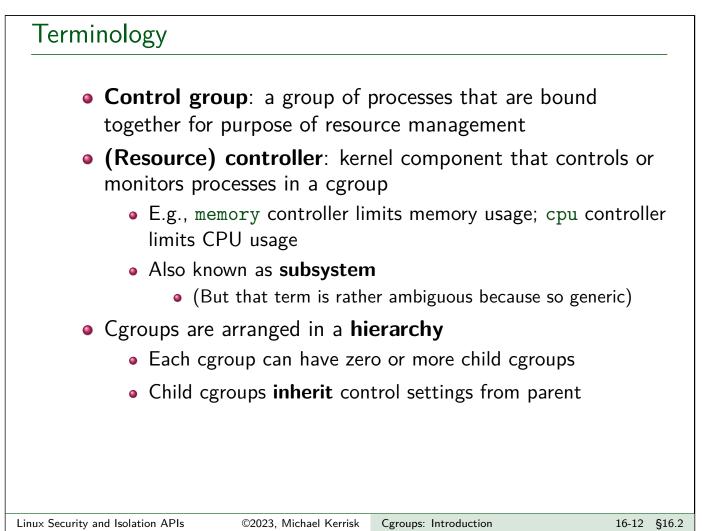
Outline

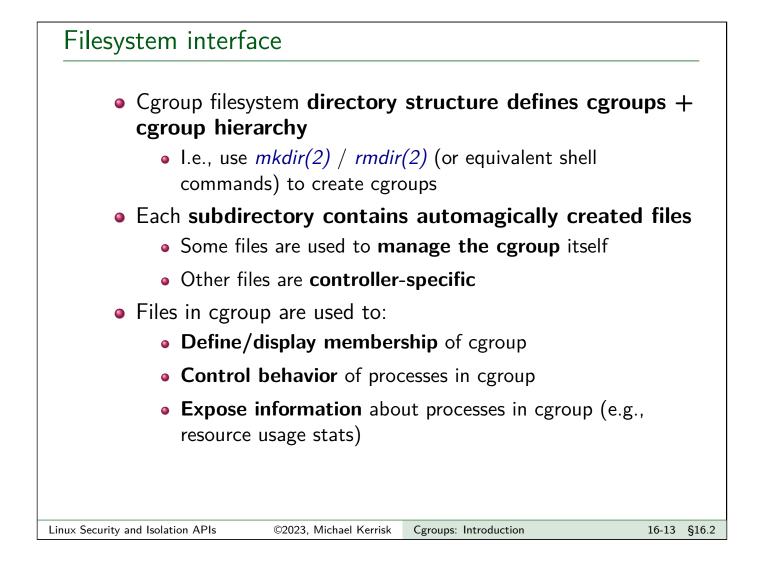
16 Cgroups: Introduction	16-1
16.1 Preamble	16-3
16.2 What are control groups?	16-9
16.3 An example: the pids controller	16-16
16.4 Creating, destroying, and populating a cgroup	16-21
16.5 Enabling and disabling controllers	16-34

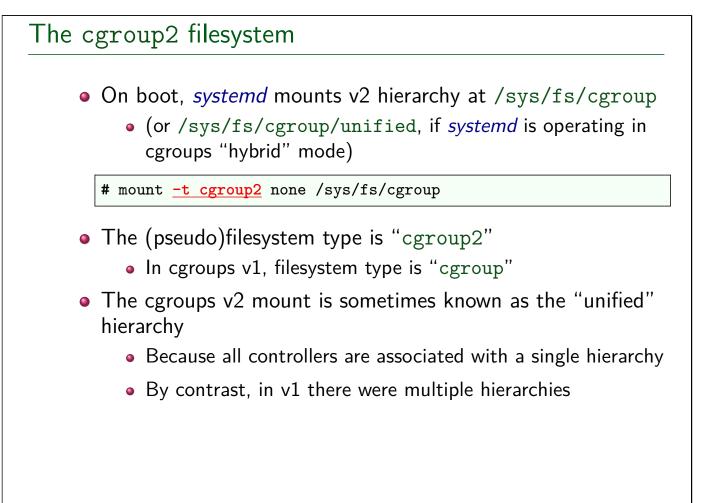
What are control groups?

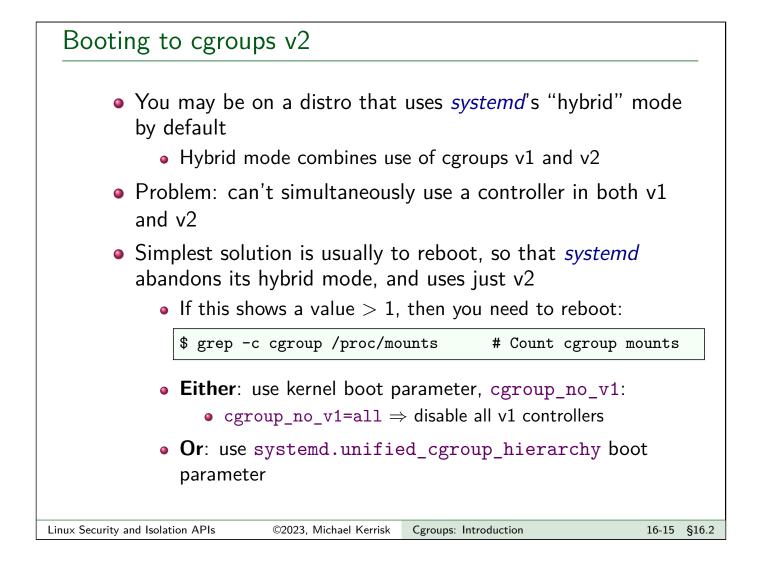
- Two principal components:
 - A mechanism for hierarchically grouping processes
 - A set of **controllers** (kernel components) that manage, control, or monitor processes in cgroups
- Interface is via a pseudo-filesystem
- Cgroup manipulation takes form of filesystem operations, which might be done:
 - Via shell commands
 - Programmatically
 - Via management daemon (e.g., *systemd*)
 - Via your container framework's tools (e.g., LXC, Docker)









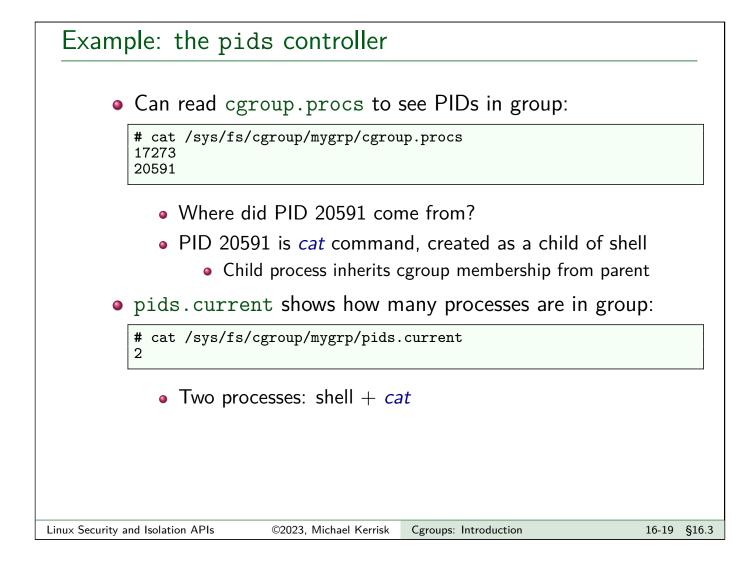


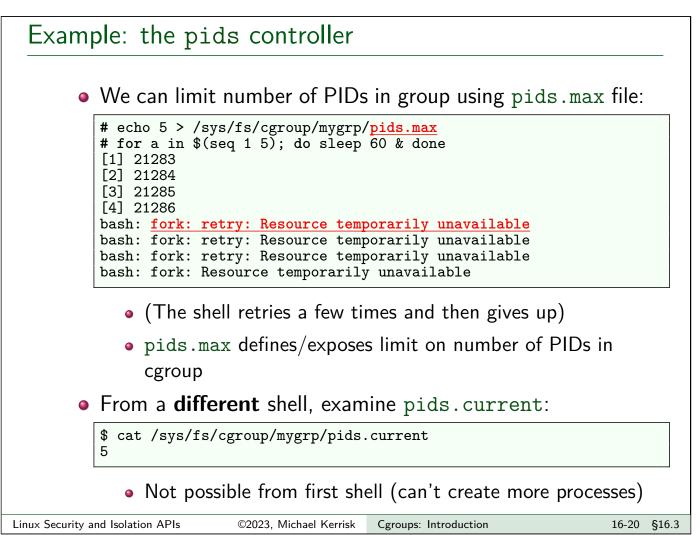
Example: the pids controller

- pids ("process number") controller allows us to limit number of PIDs in cgroup (prevent *fork()* bombs!)
- Create new cgroup, and place shell's PID in that cgroup:

```
# mkdir /sys/fs/cgroup/mygrp
# echo $$
17273
# echo $$ > /sys/fs/cgroup/mygrp/cgroup.procs
```

- cgroup.procs defines/displays PIDs in cgroup
- (Note '#' prompt \Rightarrow all commands done as superuser)
- Moving a PID into a group automatically removes it from group of which it was formerly a member
 - I.e., a process is always a member of exactly one group in the hierarchy



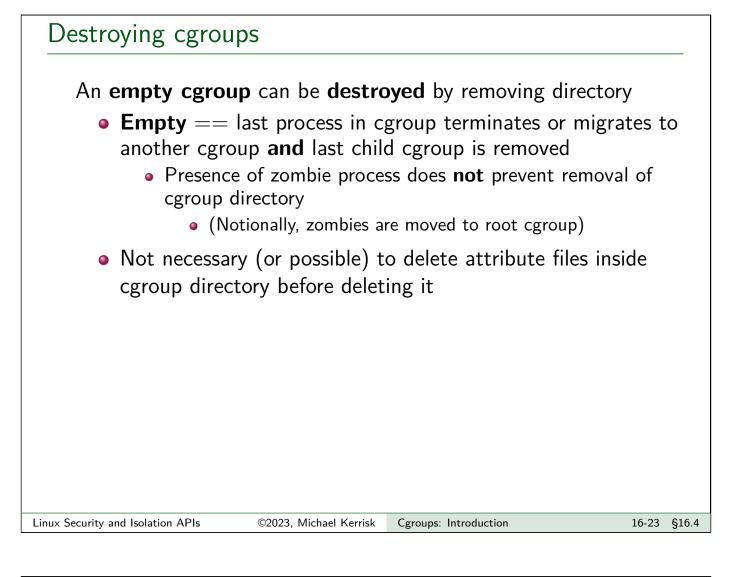


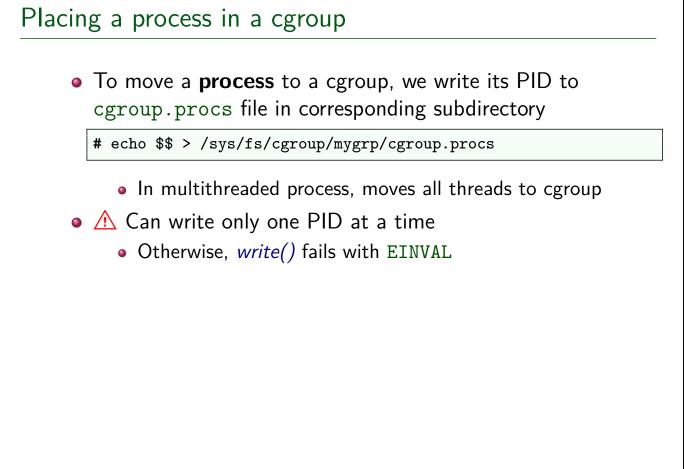
Creating cgroups

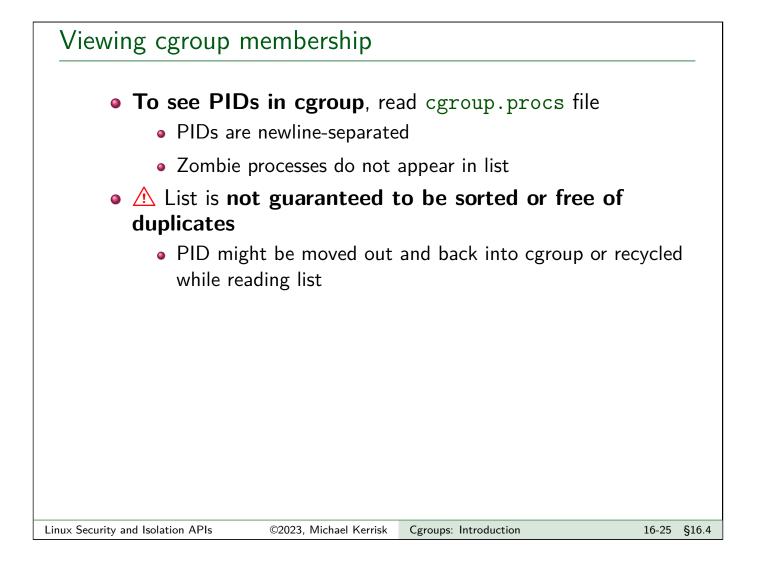
- Initially, all processes on system are members of root cgroup
- New cgroups are created by creating subdirectories under cgroup mount point:

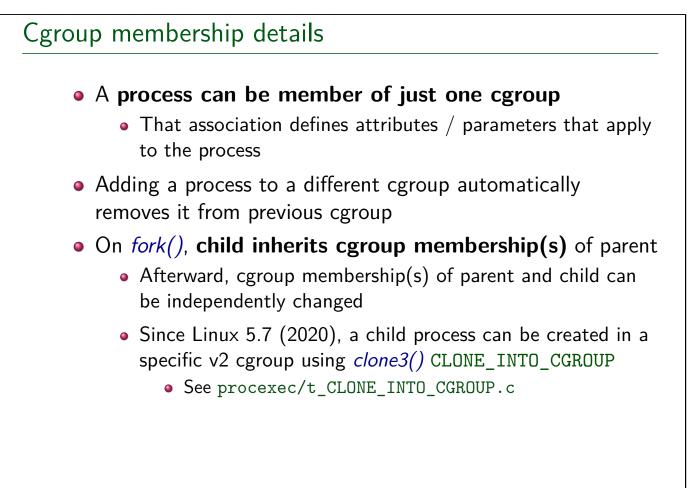
```
# mkdir /sys/fs/cgroup/mygrp
```

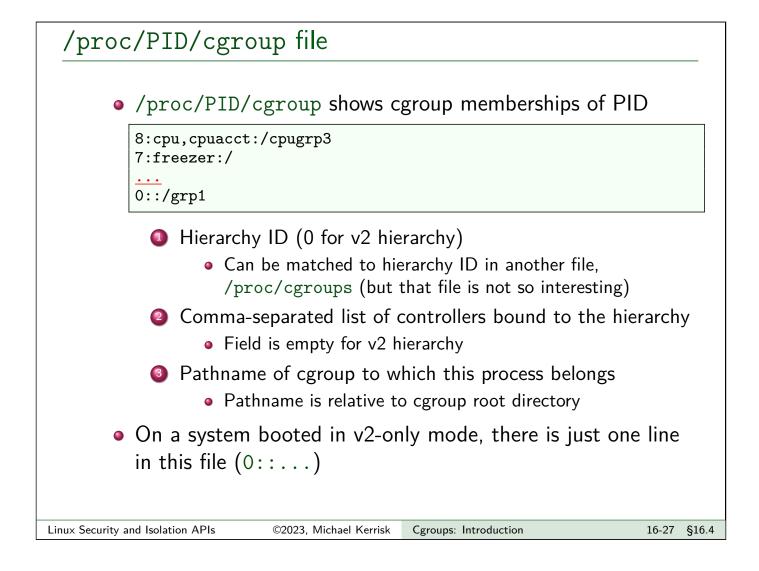
 Relationships between cgroups are reflected by creating nested (arbitrarily deep) subdirectory structure



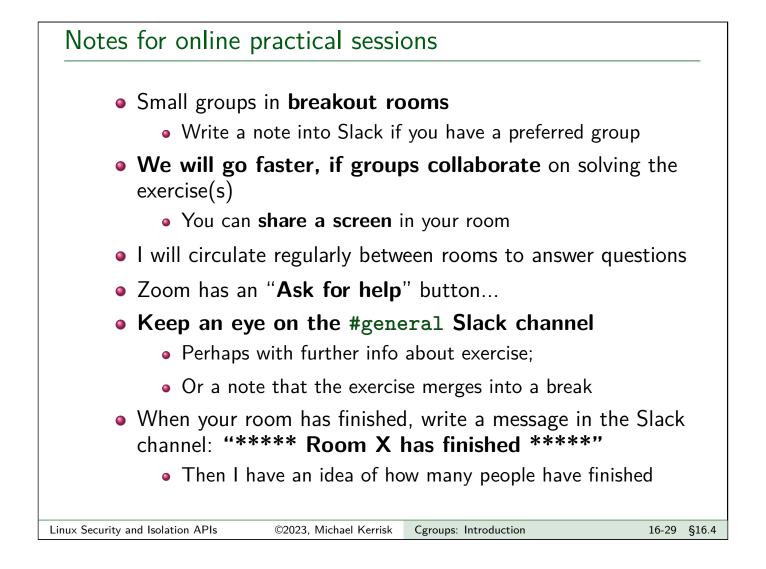


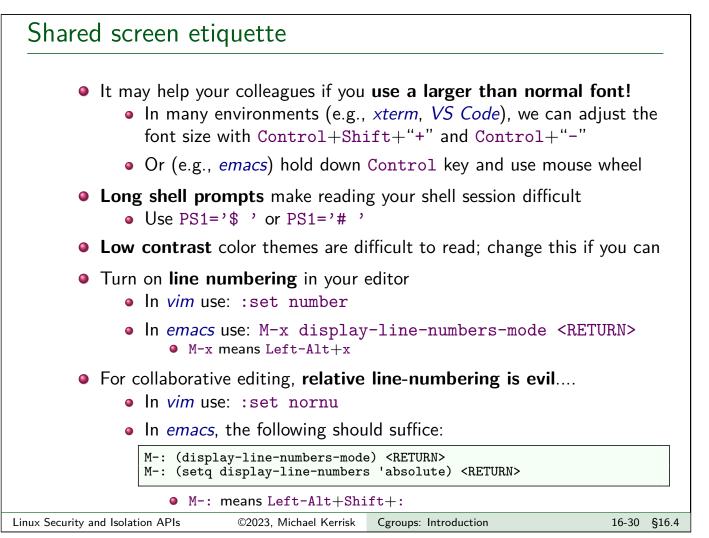


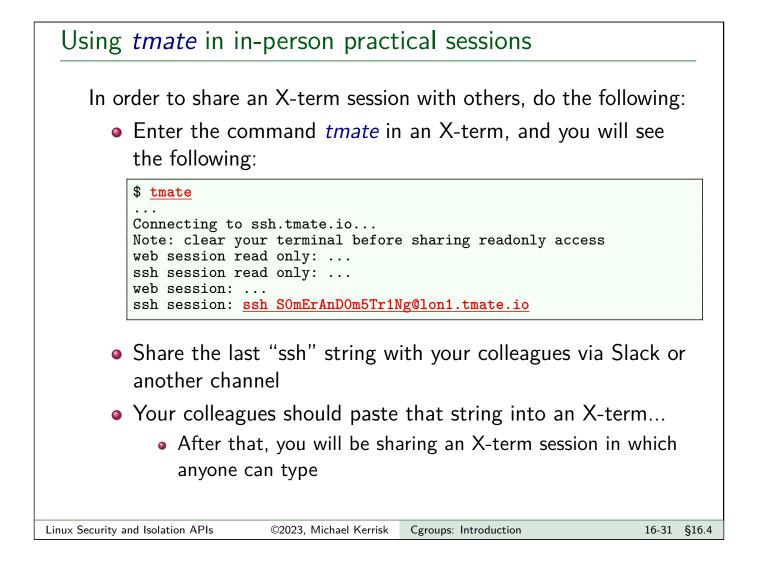




Killing all processes in a cgroup Writing "1" to cgroup.kill kills all processes in a cgroup Action is recursive I.e., processes in descendant cgroups are also killed Processes are killed using SIGKILL File is write-only, and available only in non-root cgroups :-) Available since Linux 5.14 (2021) Example use cases: Service managers (e.g., systemd) can kill all processes in a service User-space "out-of-memory" (OOM) handlers can quickly/easily kill an entire cgroup Handle some kill-container use cases that can't be handled by killing container PID 1



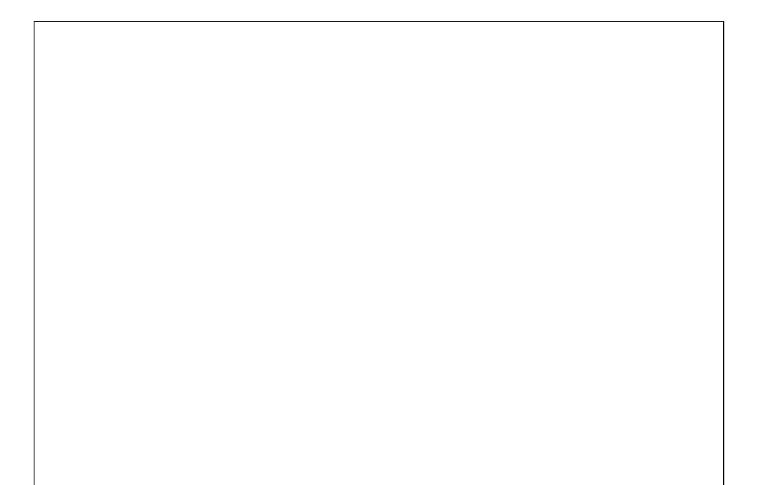




Booting to cgroups v2 • In preparation for the following exercises, if necessary reboot your system to use cgroups v2 only, as follows... • First, check whether your system is already booted to use cgroups v2 only: \$ grep cgroup /proc/mounts # Is there a v2 mount? cgroup2 /sys/fs/cgroup cgroup2 ... \$ grep cgroup /proc/mounts | grep -v name= | grep -vc cgroup2 # 0 == no v1 controllers are mounted 0 • If there is a v2 mount, and no v1 controllers are mounted, then you do not need to do anything further; otherwise: • From the GRUB boot menu, you can boot to cgroups v2-only mode by editing the boot command (select a GRUB menu entry and type "e"). In the line that begins with "linux", add the following parameter:

systemd.unified_cgroup_hierarchy

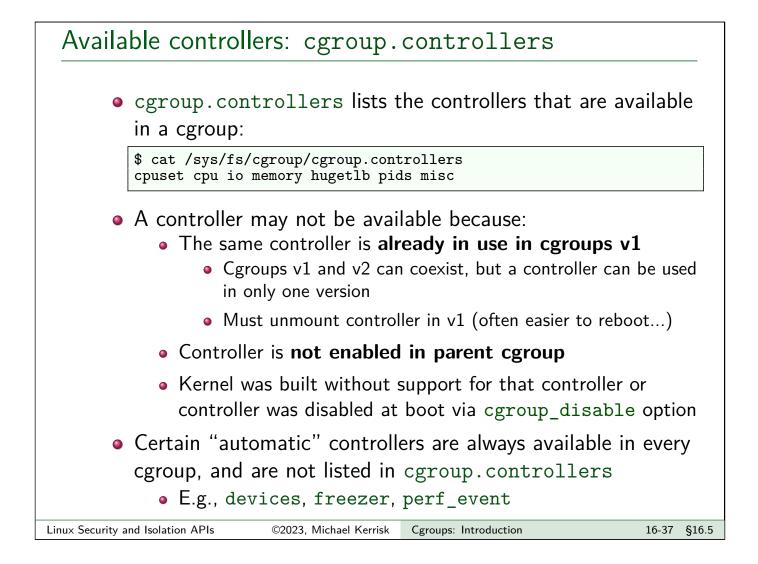
Exercises			
	e, we create a cgrou that process to a dif	p, place a process in the cg ferent cgroup.	roup, and
 Create t 	wo subdirectories, m	L and m2, in the cgroup root	directory.
	the following comm lting process:	and, and note the PID assig	gned to
# sleep 3	300 &		
	•	s created in the previous ste verify by reading the file co	•
Now wr	ite the PID of the pr	ocess into the file m2/cgrou	p.procs.
 Is the P 	ID still visible in the	<pre>file m1/cgroup.procs? Ex</pre>	plain.
	oving cgroup m1 usir this work?	ng the command rm -rf m	1. Why
	till running, kill the s m1 and m2 using the	<i>leep</i> process and then remo <i>rmdir</i> command.	ove the
Linux Security and Isolation APIs	©2023, Michael Kerrisk	Cgroups: Introduction	16-33 §16.4

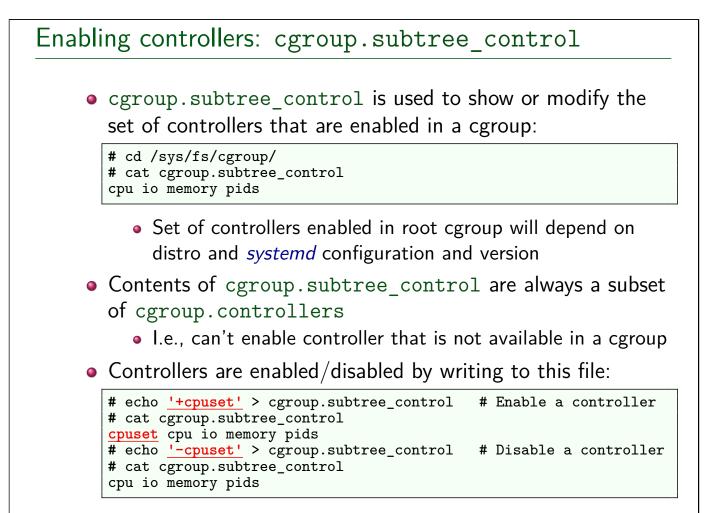


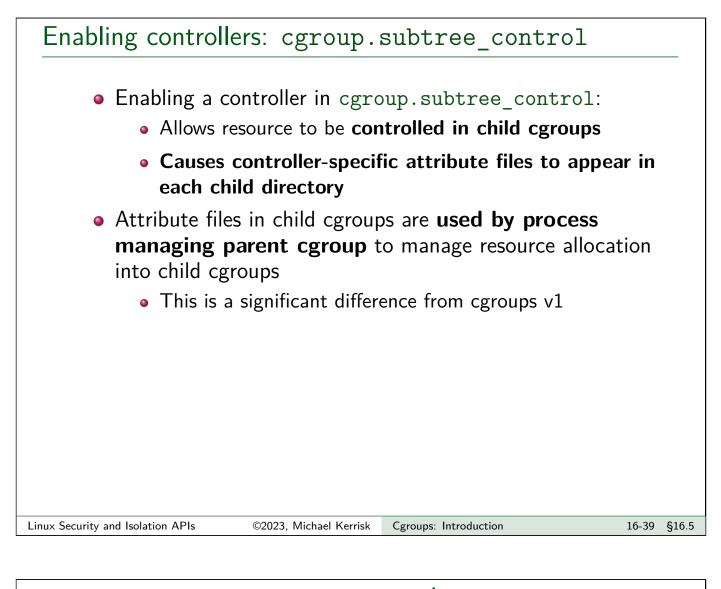
Enabling and disabling controllers

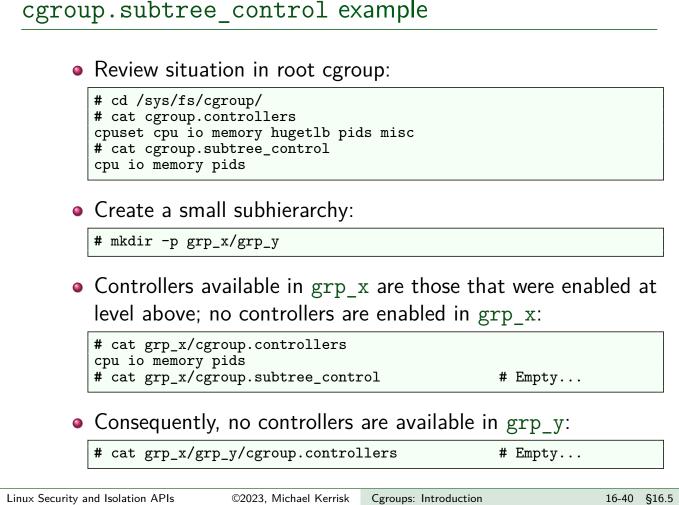
• Each cgroup v2 directory contains two files:

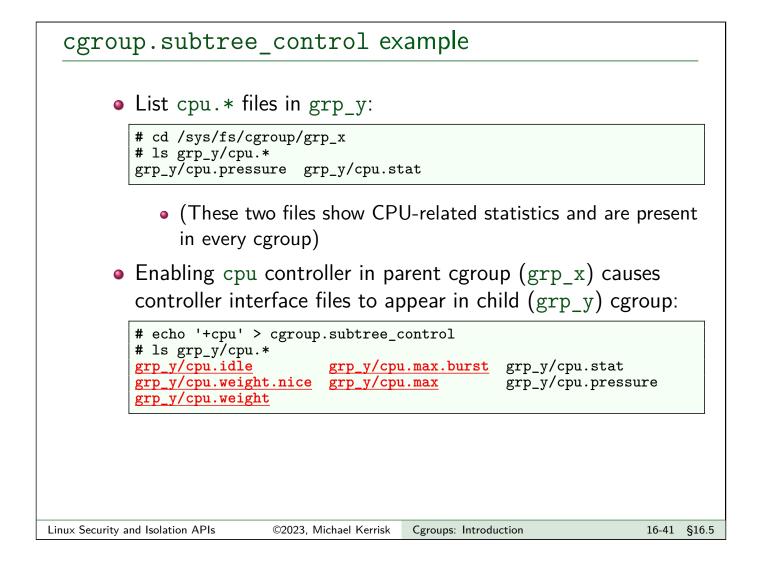
- cgroup.controllers: lists controllers that are **available** in this cgroup
- cgroup.subtree_control: used to list/modify set of controllers that are **enabled** in this cgroup
 - Always a subset of cgroup.controllers
- Together, these files allow different controllers to be managed to **different levels of granularity** in v2 hierarchy

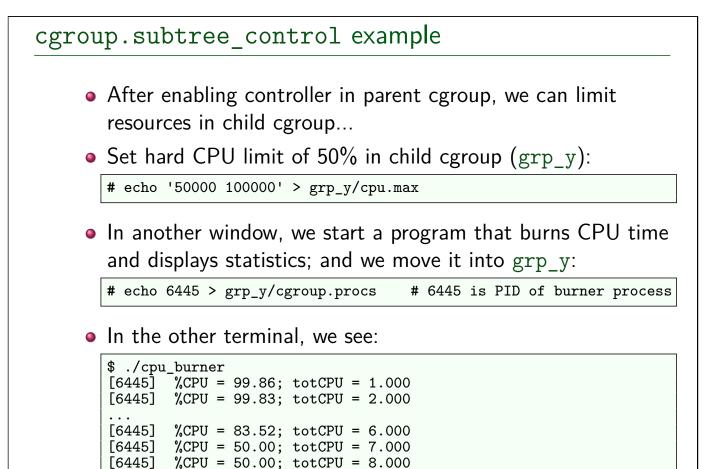




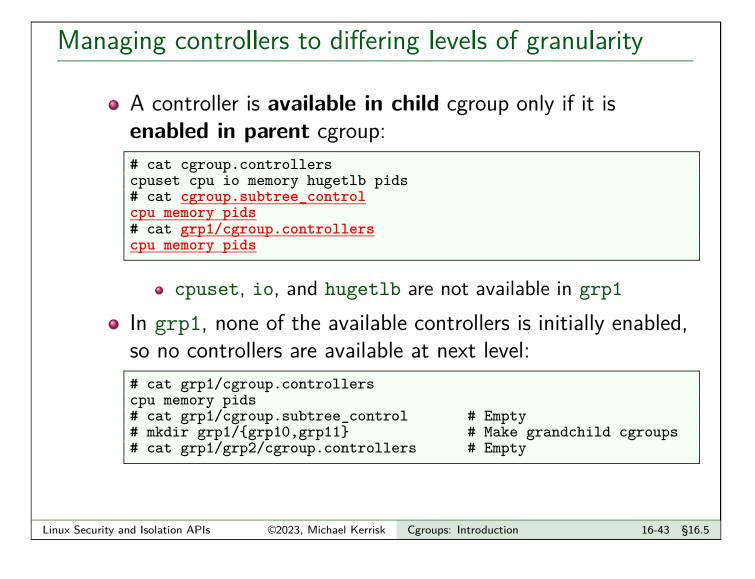


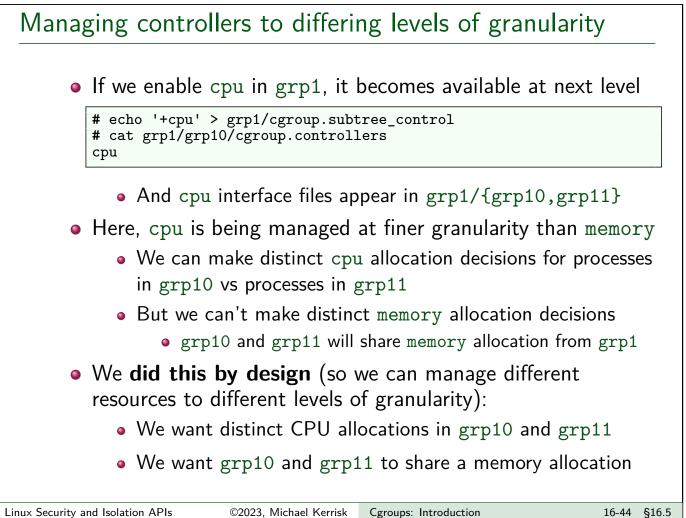






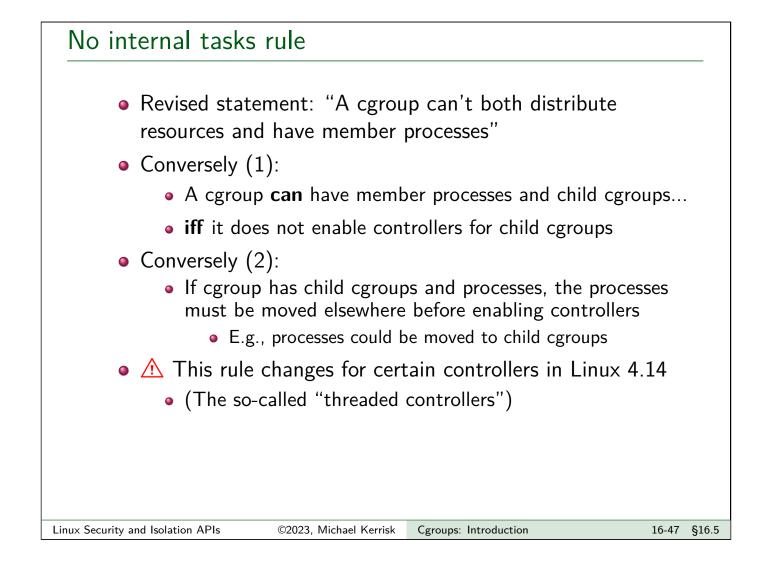
Linux Security and Isolation APIs





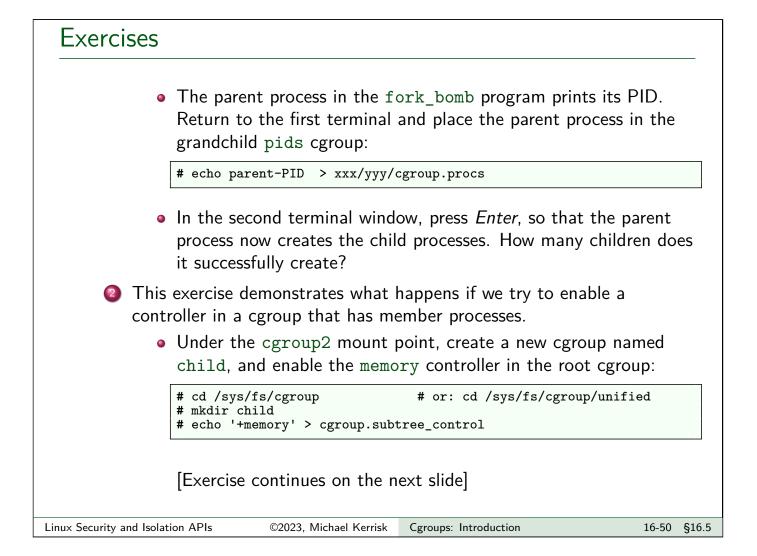
Top-down constraints • Child cgroups are always subject to any resource constraints established by controllers in ancestor cgroups • ⇒ Descendant cgroups can't relax constraints imposed by ancestor cgroups • If a controller is disabled in a cgroup (i.e., not written to cgroup.subtree_control in parent cgroup), it cannot be enabled in any descendants of the cgroup Linux Security and Isolation APIs ©2023, Michael Kerrisk Cgroups: Introduction 16-45 §16.5

No internal tasks rule Cgroups v2 enforces a rule often expressed as: "a cgroup can't have both child cgroups and member processes" I.e., only leaf nodes can have member processes The "no internal tasks" rule But the rule can be expressed more precisely... A cgroup can't both: distribute a resource to child cgroups (i.e., enable controllers in cgroup.subtree_control), and have member processes Note: root cgroup is an exception to this rule



top-o	exercise demonstrates that resource constraints apply in a down fashion, using the cgroups v2 pids controller. Check that the pids controller is visible in the cgroup root cgroup.controllers file. If it is not, reboot the kernel as described on slide 16-15.
•	To simplify the following steps, change your current directory to the cgroup root directory (i.e., the location where the cgroup2 filesystem is mounted; on recent <i>systemd</i> -based systems, this will be /sys/fs/cgroup, or possibly /sys/fs/cgroup/unified).
٩	Create a child and grandchild directory in the cgroup filesystem and enable the PIDs controller in the root directory and the first subdirectory:
	<pre># mkdir xxx # mkdir xxx/yyy # echo '+pids' > cgroup.subtree_control # echo '+pids' > xxx/cgroup.subtree_control</pre>
	[Exercise continues on next page]

Exercises • Set an upper limit of 10 tasks in the child cgroup, and an upper limit of 20 tasks in the grandchild cgroup: # echo '10' > xxx/pids.max # echo '20' > xxx/yyy/pids.max • In another terminal, use the supplied cgroups/fork bomb.c program. fork_bomb <num-children> [<child-sleep>] # Default: 0 300 Run the program with the following command line, which (after the user presses *Enter*) will cause the program to create 30 children that sleep for (the default) 300 seconds: \$./fork_bomb 30 [Exercise continues on next page...] 16-49 §16.5 Linux Security and Isolation APIs ©2023, Michael Kerrisk Cgroups: Introduction



Exercises Start a process running *sleep*, and place the process into the child cgroup: # sleep 1000 & # echo \$! > child/cgroup.procs What happens if we now try to enable the memory controller in the child cgroup via the following command? # echo '+memory' > child/cgroup.subtree_control Does the result differ if we reverse the order of the preceding steps (i.e., enable the controller, then place a process in the cgroup)?

Linux Security and Isolation APIs

©2023, Michael Kerrisk

risk Cgroups: Introduction

16-51 §16.5

Nc	otes		