Linux Security and Isolation APIs Fundamentals

Control Groups (cgroups): Introduction

Michael Kerrisk, man7.org © 2023

February 2023

mtk@man7.org

Outline	Rev: # e2bf8f005a44
11 Cgroups: Introduction	11-1
11.1 Preamble	11-3
11.2 What are control groups?	11-9
11.3 An example: the pids controller	11-16
11.4 Creating, destroying, and populating a cgroup	11-21
11.5 Enabling and disabling controllers	11-33

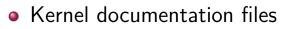
Outline11Cgroups: Introduction11-111.1 Preamble11-311.2 What are control groups?11-911.3 An example: the pids controller11-1611.4 Creating, destroying, and populating a cgroup11-2111.5 Enabling and disabling controllers11-33

Goals

• We'll focus on:

- General principles of operation; goals of cgroups
- The cgroup2 filesystem
- Interacting with cgroup2 filesystem using shell commands
- We'll look briefly at some of the controllers

Resources

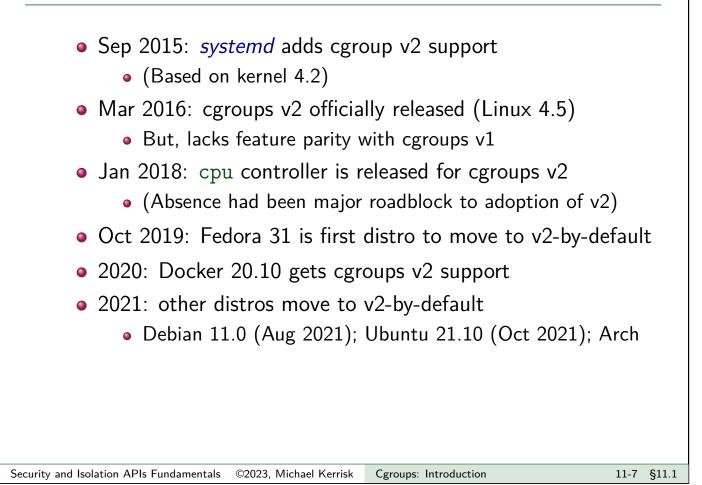


- V2: Documentation/admin-guide/cgroup-v2.rst
- V1: Documentation/admin-guide/cgroup-v1/*.rst
 - Before Linux 5.3: Documentation/cgroup-v1/*.txt
- cgroups(7) man page
- Neil Brown's (2014) LWN.net series on cgroups: https://lwn.net/Articles/604609/
 - Thought-provoking commentary on the meaning of grouping and hierarchy
- https://lwn.net/Articles/484254/ Tejun Heo's initial thoughts about redesigning cgroups (Feb 2012)
 - See also https://lwn.net/Articles/484251/, *Fixing Control Groups*, Jon Corbet, Feb 2012
- Other articles at https://lwn.net/Kernel/Index/#Control_groups

```
Security and Isolation APIs Fundamentals ©2023, Michael Kerrisk Cgroups: Introduction 11-5 §11.1
```

Some history 2006/2007, "Process Containers" @ Google ⇒ Cgroups v1 Jan 2008: initial mainline kernel release (Linux 2.6.24) Three resource controllers (all CPU-related) in initial release Subsequently, other controllers are added memory, devices, freezer, net_cls, blkio... But a few years of uncoordinated design leads to a mess Decentralized design fails us... again 2012: work has already begun on cgroups v2...

Some history



We have passed the tipping point

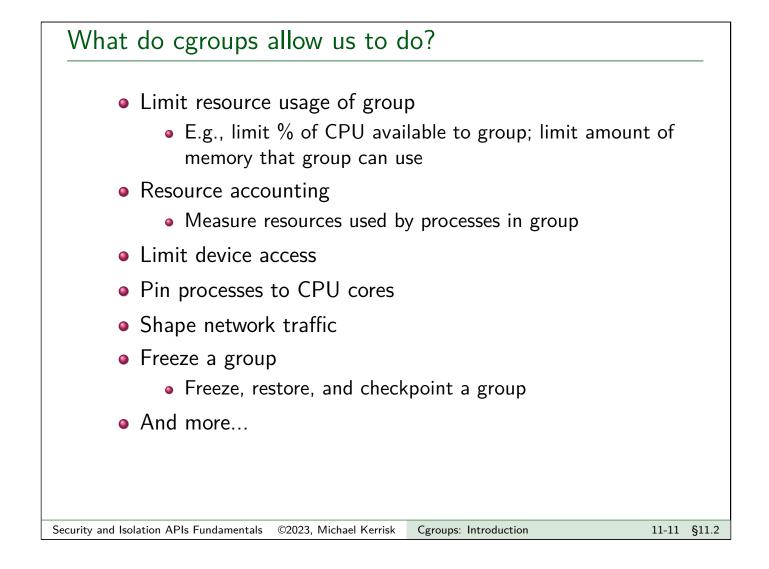
- Various (proprietary) infrastructure still depends on cgroups v1
- But:
 - A lot of migration work has already been done, *systemd* supports pure v2-only, and the distros have migrated to v2
 - Cgroups v2 offers a number of advantages over v1
- ullet \Rightarrow we'll focus on cgroups v2, and largely ignore cgroups v1

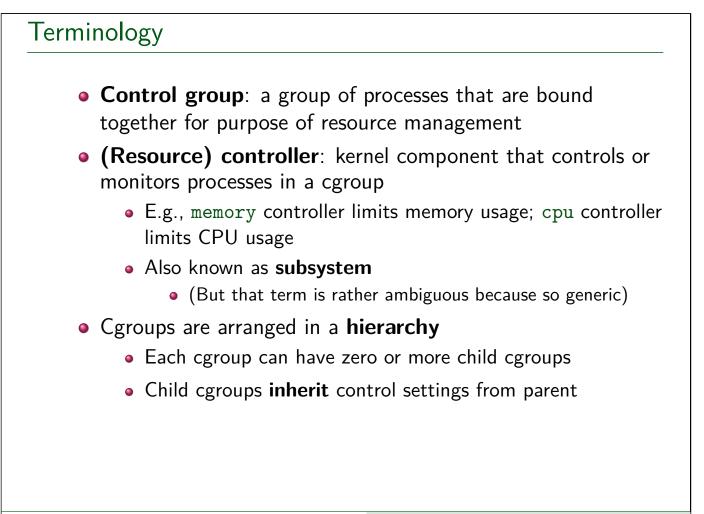
Outline

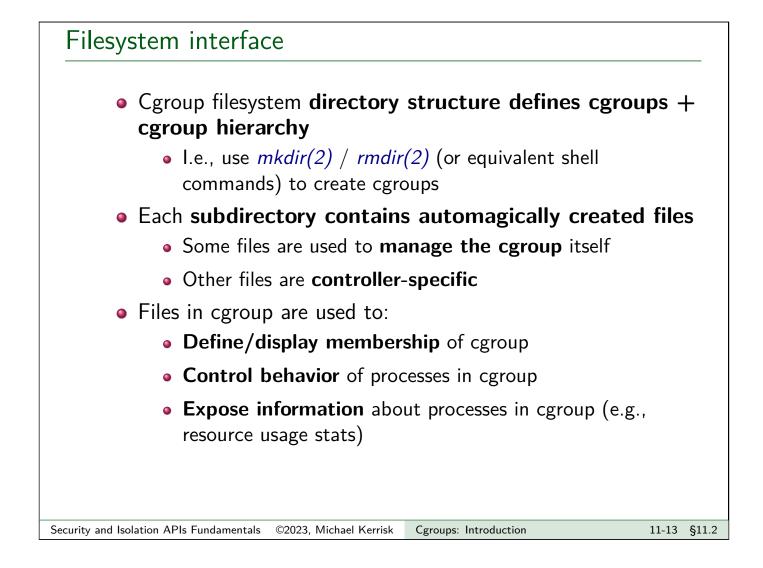
11 Cgroups: Introduction	11-1
11.1 Preamble	11-3
11.2 What are control groups?	11-9
11.3 An example: the pids controller	11-16
11.4 Creating, destroying, and populating a cgroup	11-21
11.5 Enabling and disabling controllers	11-33

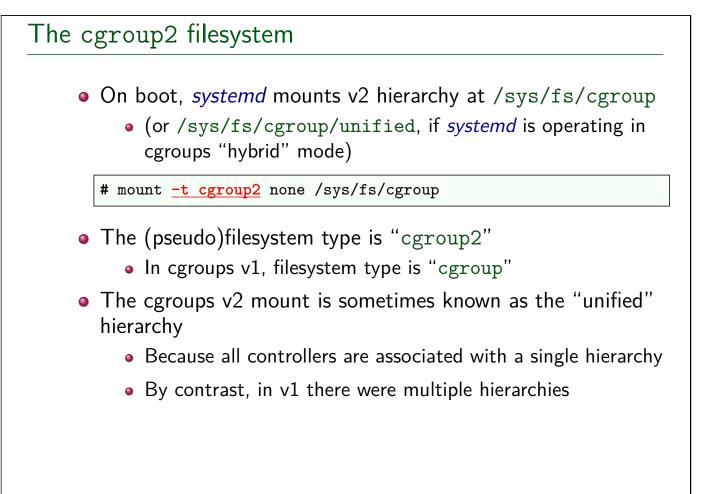
What are control groups?

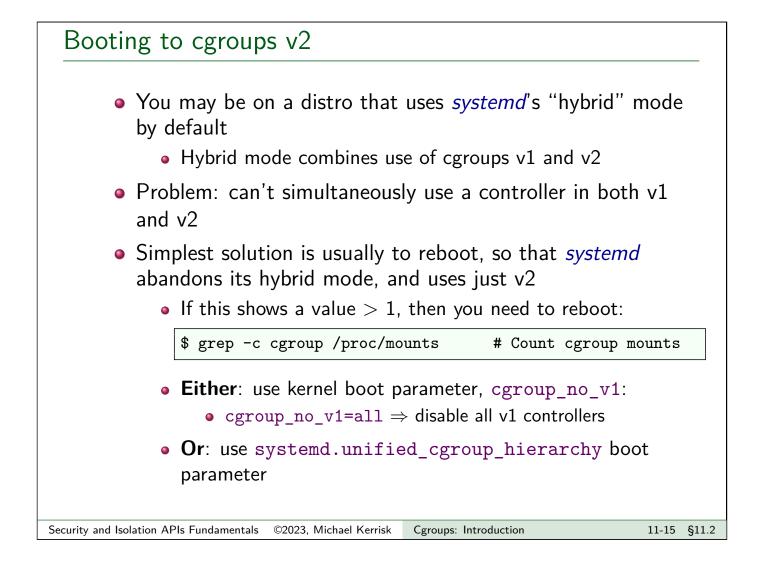
- Two principal components:
 - A mechanism for hierarchically grouping processes
 - A set of **controllers** (kernel components) that manage, control, or monitor processes in cgroups
- Interface is via a pseudo-filesystem
- Cgroup manipulation takes form of filesystem operations, which might be done: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











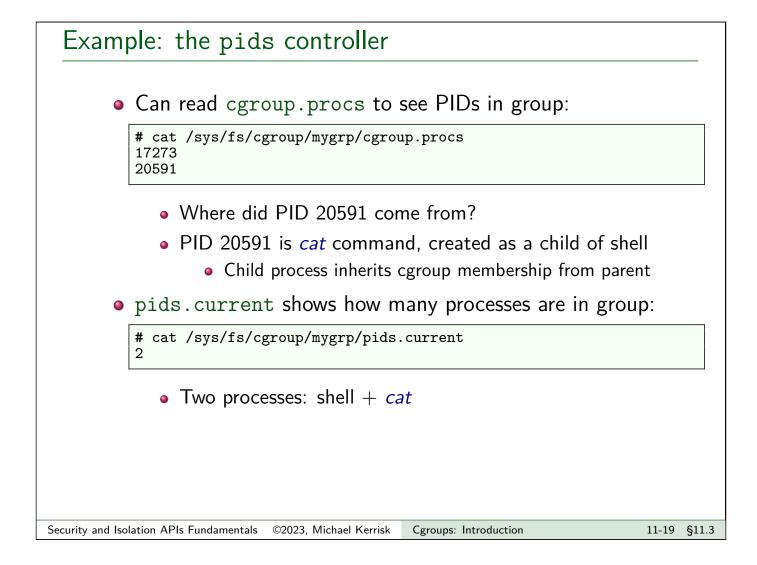
Outline11Cgroups: Introduction11-111.1Preamble11-311.2What are control groups?11-911.3An example: the pids controller11-1611.4Creating, destroying, and populating a cgroup11-2111.5Enabling and disabling controllers11-33

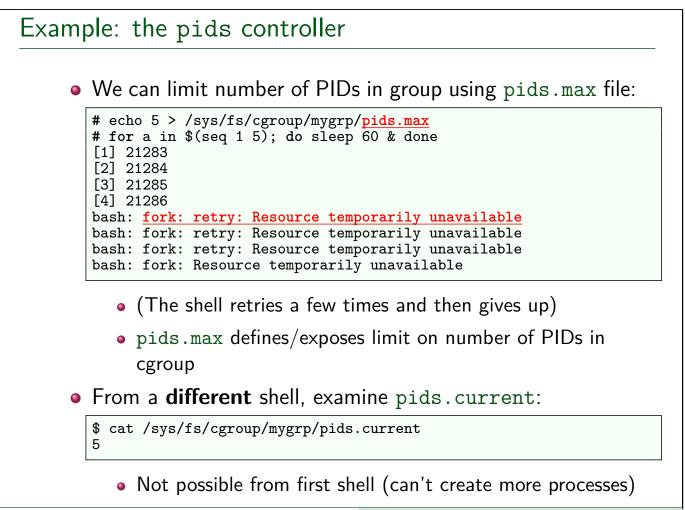
Example: the pids controller

- 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





11-20 §11.3

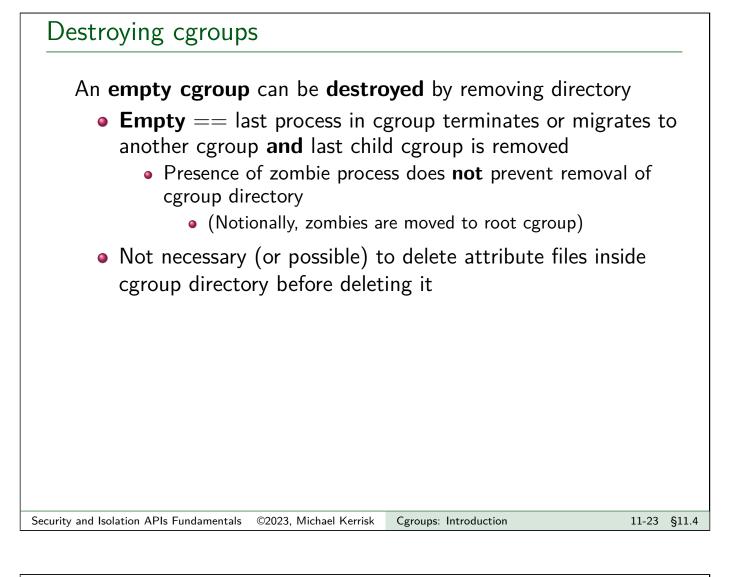
Outline11Cgroups: Introduction11-111.1Preamble11-311.2What are control groups?11-911.3An example: the pids controller11-1611.4Creating, destroying, and populating a cgroup11-2111.5Enabling and disabling controllers11-33

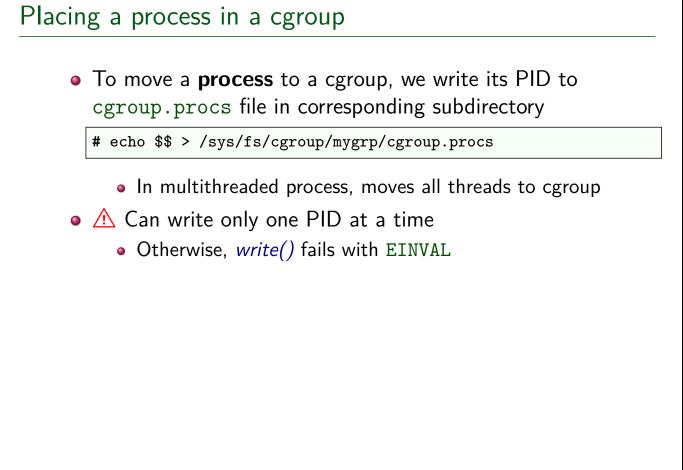
Creating cgroups

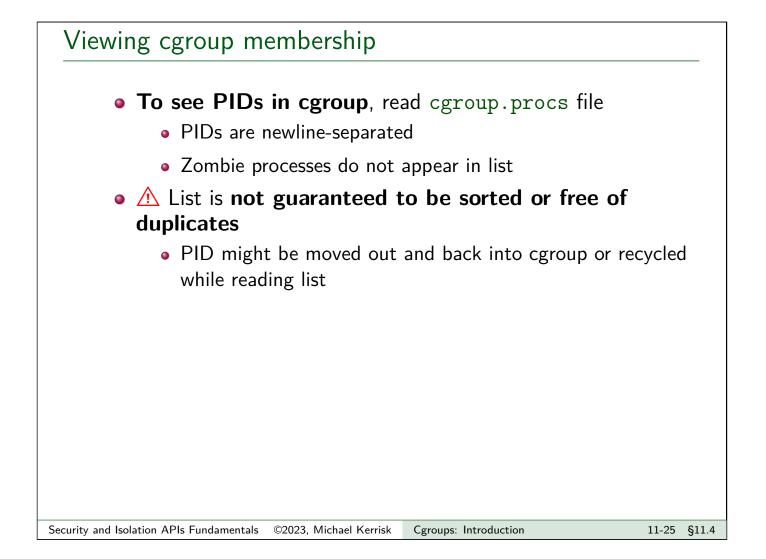
- Initially, all processes on system are members of root cgroup
- New cgroups are created by creating subdirectories under cgroup mount point:

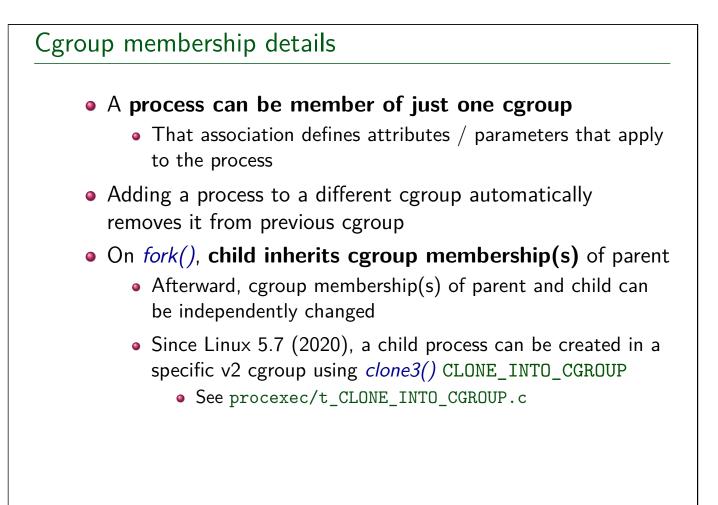
```
# mkdir /sys/fs/cgroup/mygrp
```

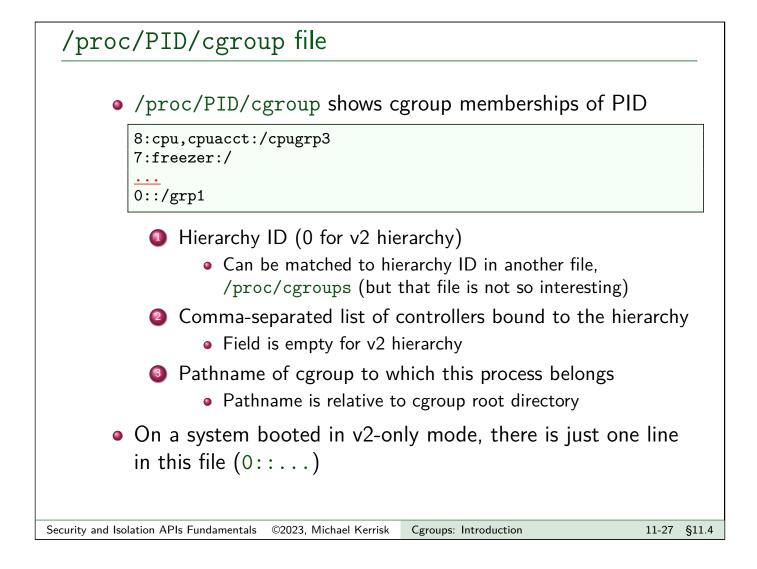
 Relationships between cgroups are reflected by creating nested (arbitrarily deep) subdirectory structure

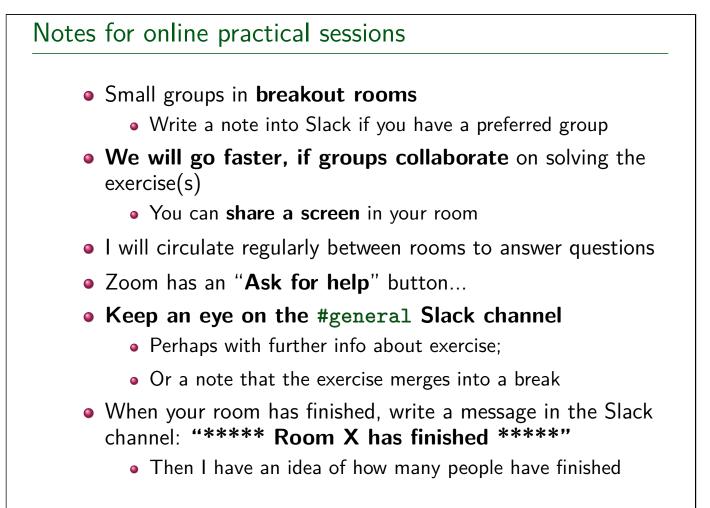


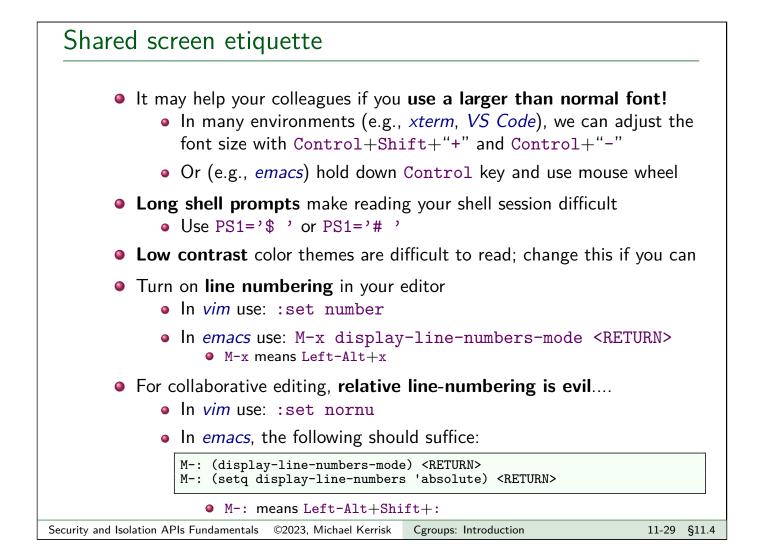












Using tmate in in-person practical sessions In order to share an X-term session with others, do the following: Enter the command tmate in an X-term, and you will see the following: \$ tmate Connecting to ssh.tmate.io... Note: clear your terminal before sharing readonly access web session read only: ... ssh session read only: ... ssh session: ssh SOMErAnDOm5Tr1Ng@lon1.tmate.io Share the last "ssh" string with your colleagues via Slack or another channel

- Your colleagues should paste that string into an X-term...
 - After that, you will be sharing an X-term session in which anyone can type

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 # 0 == no v1 controllers are mounted • 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 11-31 §11.4 Security and Isolation APIs Fundamentals ©2023, Michael Kerrisk Cgroups: Introduction

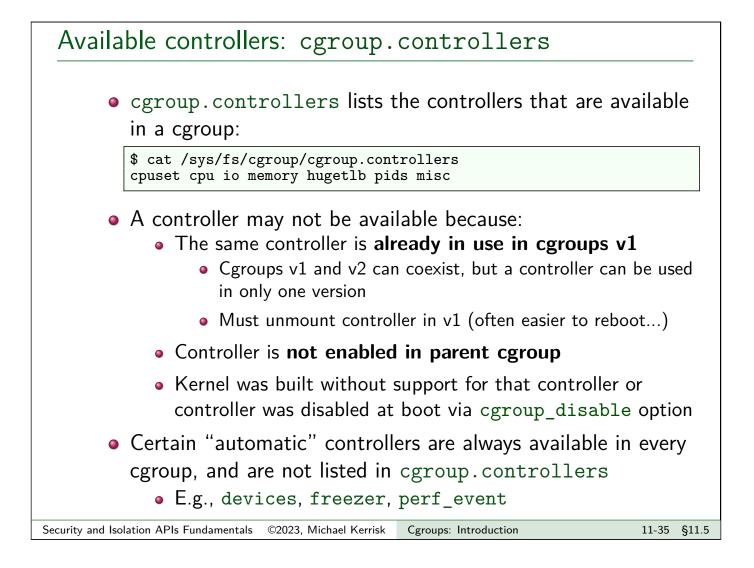
Exercises			
	is exercise, we create a cgrou migrate that process to a dif	p, place a process in the cgro ferent cgroup.	up, and
۹	Create two subdirectories, m	1 and m2, in the cgroup root d	lirectory.
٩	Execute the following comm the resulting process:	and, and note the PID assign	ed to
	# sleep 300 &		
٩	•	s created in the previous step verify by reading the file cont	
٥	Now write the PID of the pr	ocess into the file m2/cgroup	.procs.
٩	Is the PID still visible in the	<pre>file m1/cgroup.procs? Exp</pre>	lain.
٩	Try removing cgroup m1 usin doesn't this work?	ng the command rm -rf m1.	Why
•	If it is still running, kill the s cgroups m1 and m2 using the	<i>sleep</i> process and then remove e <i>rmdir</i> command.	e the
curity and Isolation A	Pls Fundamentals ©2023, Michael Kerrisk	Cgroups: Introduction	11-32 §11.

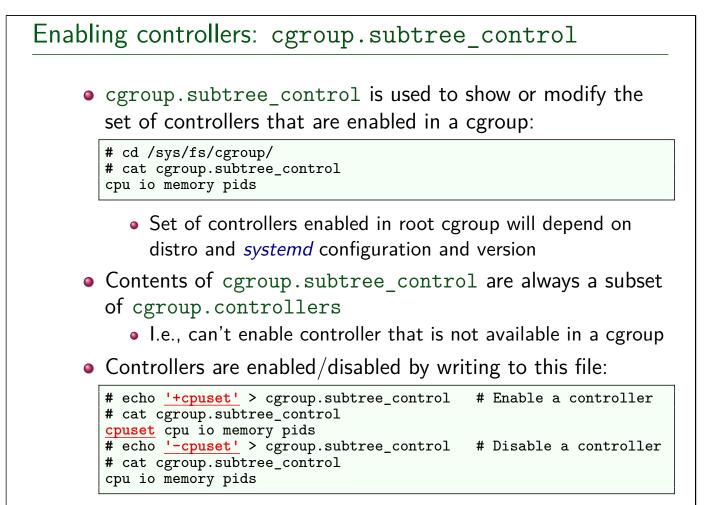
Outline11 Cgroups: Introduction11-111.1 Preamble11-311.2 What are control groups?11-911.3 An example: the pids controller11-1611.4 Creating, destroying, and populating a cgroup11-2111.5 Enabling and disabling controllers11-33

Enabling and disabling controllers

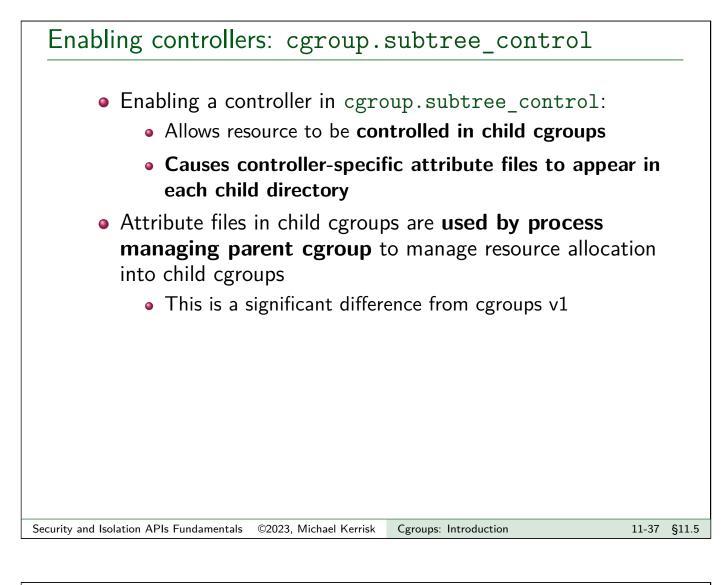
• Each cgroup v2 directory contains two files:

- 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

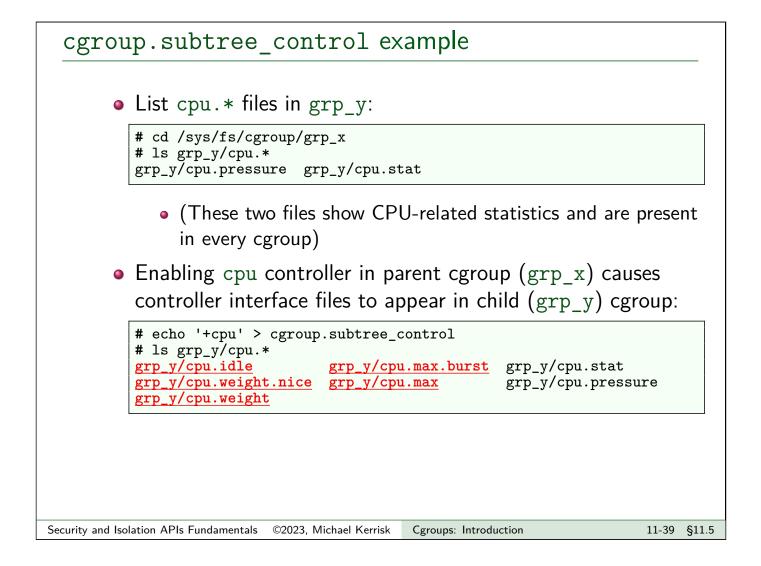


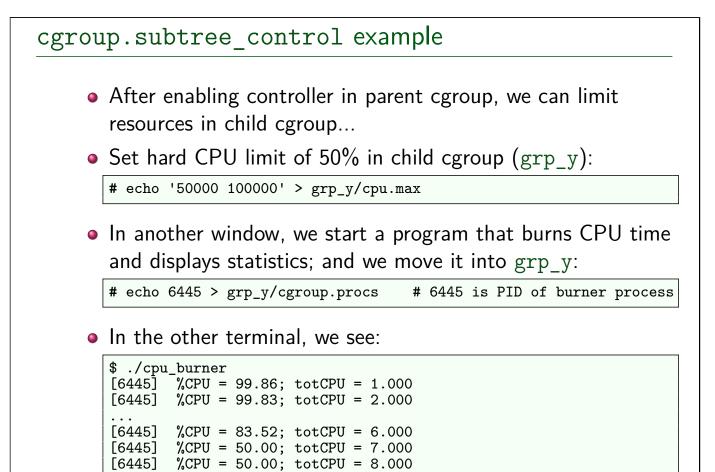


11-36 §11.5



۲	Review situation in root cgroup:
	<pre># cd /sys/fs/cgroup/ # cat cgroup.controllers cpuset cpu io memory hugetlb pids misc # cat cgroup.subtree_control cpu io memory pids</pre>
۲	Create a small subhierarchy:
	<pre># mkdir -p grp_x/grp_y</pre>
۲	Controllers available in grp_x are those that were enabled at level above; no controllers are enabled in grp_x :
	<pre># cat grp_x/cgroup.controllers cpu io memory pids # cat grp_x/cgroup.subtree_control</pre>
•	Consequently, no controllers are available in grp_y:
	<pre># cat grp_x/grp_y/cgroup.controllers # Empty</pre>

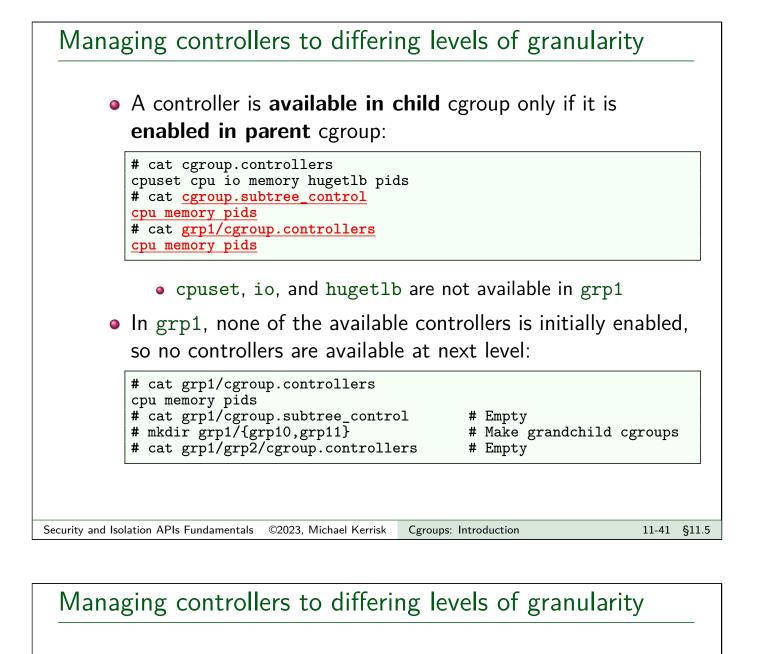




```
Security and Isolation APIs Fundamentals ©2023, Michael Kerrisk Cgroups: Introduction
```

. . .

11-40 §11.5



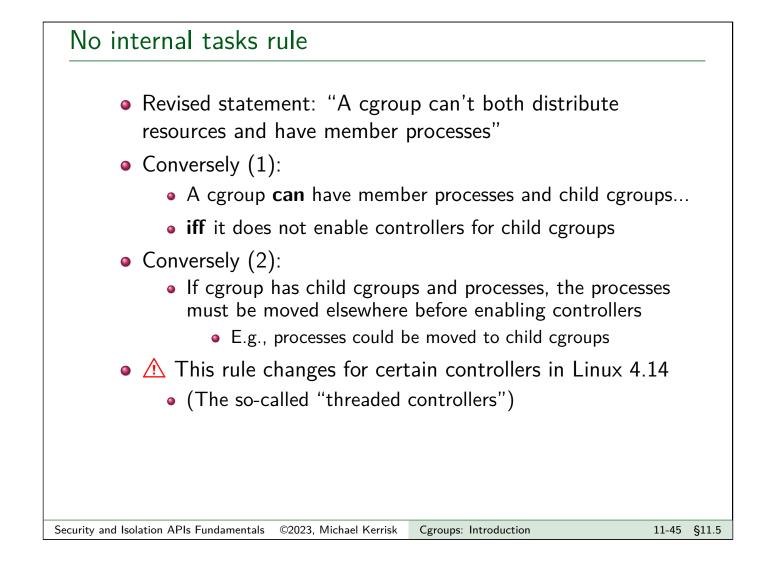
• If we enable cpu in grp1, it becomes available at next level

```
# echo '+cpu' > grp1/cgroup.subtree_control
# cat grp1/grp10/cgroup.controllers
cpu
```

- And cpu interface files appear in grp1/{grp10,grp11}
- Here, cpu is being managed at finer granularity than memory
 - We can make distinct cpu allocation decisions for processes in grp10 vs processes in grp11
 - But we can't make distinct memory allocation decisions
 - grp10 and grp11 will share memory allocation from grp1
- We **did this by design** (so we can manage different resources to different levels of granularity):
 - We want distinct CPU allocations in grp10 and grp11
 - We want grp10 and grp11 to share a memory allocation

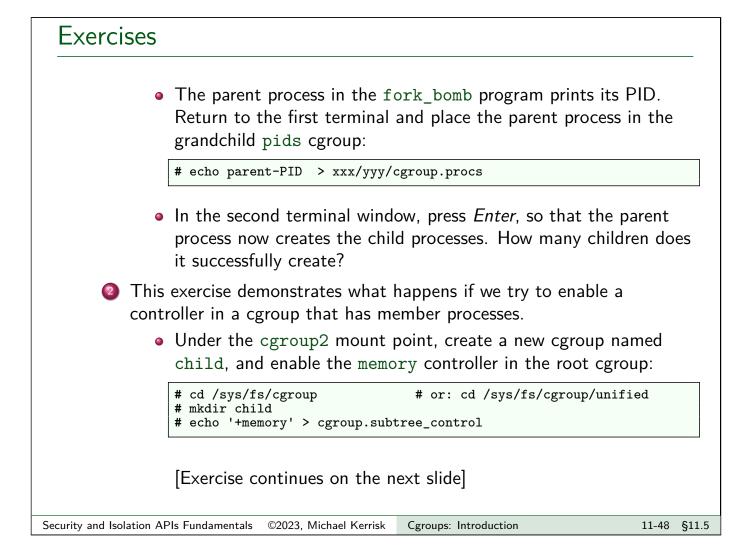
Top-down constraints • Child cgroups are always subject to any resource constraints established by controllers in ancestor cgroups • ⇒ Descendant cgroups can't relax constraints imposed by ancestor cgroups • If a controller is disabled in a cgroup (i.e., not written to cgroup.subtree_control in parent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

No internal tasks rule Cgroups v2 enforces a rule often expressed as: "a cgroup can't have both child cgroups and member processes" l.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



 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 systemd-based systems, this will be /sys/fs/cgroup, or possibly /sys/fs/cgroup/unified). 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</pre>	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 11-15.
<pre>and enable the PIDs controller in the root directory and the first subdirectory: # mkdir xxx # mkdir xxx/yyy # echo '+pids' > cgroup.subtree_control</pre>	۹	the cgroup root directory (i.e., the location where the cgroup2 filesystem is mounted; on recent <i>systemd</i> -based systems, this will
<pre># mkdir xxx/yyy # echo '+pids' > cgroup.subtree_control</pre>	٩	and enable the PIDs controller in the root directory and the first
		<pre># mkdir xxx/yyy # echo '+pids' > cgroup.subtree_control</pre>

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...] 11-47 §11.5 Security and Isolation APIs Fundamentals ©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)?

Security and Isolation APIs Fundamentals $$\sc @2023$$, Michael Kerrisk

risk Cgroups: Introduction

11-49 §11.5

Notes		