Agenda
Agenda

- What are cgroups?
- Why use cgroups?
- How is cgroups implemented?
  - Subsystems
  - cgroup filesystem
  - cgroup hierarchy
cgroup filesystem

Overview cgroups Subsystems
- Group CPU Scheduler
- CPU Accounting Controller
- Cpuset
- Memory
- Block IO Controller
- Device Whitelist Controller
- Freezer
- Namespace
Agenda

- libcgroup
- Exercises / Demonstration of various cgroups setups
What Are Cgroups?
What Are Cgroups?

- Control Groups
- generic process-grouping framework
- in Linux Kernel (since 2.6.24)
- CONFIG_CGROUPS
task  Userspace or kernel process

cgroup  One or more tasks

subsystem  Module to modify the behavior of the tasks in a cgroup

hierarchy  Several cgroups in a tree
Why Use Cgroups?
Why Use Cgroups?

How to Control the Vast Amount of Resources of Today’s Platforms?

- CPUs have multiple cores, usually machines are SMP platforms
- "many cores"
- More and more memory
Why Use Cgroups?

How to Control Resources?

- Virtual Machines
- Containers
- ... what about the native Operating System? Linux?!
Why Use Cgroups?

How to Control Resources in Operating Systems with Many Tasks?

- on "many cores"?
- with lots of memory?
Figure: Grouping Example of a University System
Figure: Hierarchy Grouping Example
Subsystems in a Group

Root Group

CPU Subsystem

Memory Subsystem

Figure: Two Subsystems in a Group
Figure: The Same Set of Subsystems Is Inherited By All Children
Different Set of Subsystems

Figure: Two Different Hierarchies to Get Different Subsystems
How Is Cgroups Implemented?
Virtual File System: cgroup
Virtual File System: cgroup

- Virtual File System cgroup
  - userspace access
  - a cgroup is a directory
  - lists tasks per cgroup

- Modification in Kernel Syscalls
  - exit()
  - fork()
  - ...

© B1 Systems GmbH 2006 – 2010
Chapter 3, Slide 19
Cgroup Subsystems
Subsystems get enabled as a mount option of the cgroup file system:

- `mount -t cgroup -o$ subsystem nodev /dev/cgroup`

Enabled subsystems spawn files in each cgroup (directory):

- `/dev/cgroup/professors/subsysA.optionB`

Overview in proc-filesystem: `/proc/cgroups`

(Overview in kernel-source:
`/usr/src/linux/include/linux/cgroup_subsys.h`
Cgroup File System
# mkdir /dev/cgroup
# mount -tcgroup xxx /dev/cgroup/
# ls /dev/cgroup/
cpu.shares
cpuacct.usage
cpuset.cpu_exclusive
cpuset.cpus
[...]
notify_on_release
release_agent
tasks
# mount
[...]
xxx on /dev/cgroup type cgroup (rw)
# umount xxx
Creating a Cgroup

~ # cd /dev/cgroup/
/dev/cgroup # mkdir professors
/dev/cgroup # cd professors/
/dev/cgroup/professors # ls
[...]
notify_on_release
tasks
/dev/cgroup/professors # wc -l tasks
0 tasks
/dev/cgroup/professors # 
/dev/cgroup/professors # wc -l ../tasks
142 ../tasks
/dev/cgroup/professors #
Deleting a Cgroup

/dev/cgroup # rm professors/
rm: cannot remove ‘professors/’: Is a directory
/dev/cgroup # rm -rf professors/
[...]
rm: cannot remove ‘professors/cpuset.cpus’: Operation not permitted
rm: cannot remove ‘professors/notify_on_release’: Operation not permitted
rm: cannot remove ‘professors/tasks’: Operation not permitted
/dev/cgroup # rmdir professors/
/dev/cgroup # echo $? 
0
/dev/cgroup #
Cgroup Default Options

# ls /dev/cgroup/
[...] notify_on_release
release_agent
tasks
# cat /dev/cgroup/notify_on_release
0
# cat /dev/cgroup/release_agent

# cat /dev/cgroup/tasks
1
[...] 3356
3457
#

~ # mount -tcgroup -ocpu,devices yyy /dev/cgroup
~ # cd /dev/cgroup/
/dev/cgroup # ls -1
  cpu.shares
devices.allow
devices.deny
devices.list
  notify_on_release
  release_agent
tasks
/dev/cgroup # mount
[...]
  yyy on /dev/cgroup type cgroup (rw,cpu,devices)
/dev/cgroup #
Add Subsystems

/dev/cgroup # mount
[...]
yyy on /dev/cgroup type cgroup (rw,cpu,devices)
/dev/cgroup # mount -oremount,cpuacct /dev/cgroup
/dev/cgroup # ls -l
 cpu.shares
 cpuacct.usage
 devices.allow
[...]
 notify_on_release
 release_agent
 tasks
/dev/cgroup # mount
[...]
yyy on /dev/cgroup type cgroup (rw,cpu,devices,cpuacct)
Attaching Processes

/dev/cgroup/professors # echo $$ > tasks
/dev/cgroup/professors # cat tasks
3356
3744
/dev/cgroup/professors # echo $$
3356
/dev/cgroup/professors # grep $$ ..:/tasks
/dev/cgroup/professors # cd ..
/dev/cgroup # rmdir professors/
rmdir: failed to remove ‘professors/’: Device or resource busy
/dev/cgroup # echo $$ > tasks
/dev/cgroup # rmdir professors/
/dev/cgroup # echo $? 
0
/dev/cgroup #
Cgroup Subsystems
Generic Overview

To get an overview of available (enabled & disabled) subsystems and their subsystem name run cat /proc/cgroups

```
~ # cat /proc/cgroups
#subsys_name hierarchy num_cgroups enabled
 cpuset  0   1   1
    ns  0   1   1
   cpu  0   1   1
cpuacct 0   1   1
  memory 0   1   0
   devices 0   1   1
   freezer 0   1   1
~ #
```

Disable subsystems: cgroup_disable=subsystem1 [,subsystem2] (Kernel Parameter)
Subsystem: Group CPU Scheduler
~ # mount -t cgroup -o cpu cpu_example /dev/cggroup/
~ # cd /dev/cggroup/
/dev/cggroup # ls
   cpu.shares notify_on_release release_agent tasks
/dev/cggroup # cat cpu.shares
1024
/dev/cggroup # mount
[...] 
cpu_example on /dev/cggroup type cgroup (rw,cpu)
/dev/cggroup #
Depending on the Kernel configuration the cgroup cpu subsystems does not allow all types of tasks:

- `CONFIG_FAIR_GROUP_SCHED=y`
  - RT-tasks not supported for grouping
- `CONFIG_RT_GROUP_SCHED=y`
  - only accepts RT-tasks if there is a way to run them
/dev/cgroup # mkdir low high
/dev/cgroup # echo 512 > low/cpu.shares
/dev/cgroup # echo 2048 > high/cpu.shares
/dev/cgroup # yes low > /dev/null &
[1] 440
/dev/cgroup # echo $! > low/tasks
/dev/cgroup # yes high > /dev/null &
[2] 523
/dev/cgroup # echo $! > high/tasks
/dev/cgroup # ps -C yes -o pid,%cpu,psr,args
  PID %CPU PSR COMMAND
  440 81.2 0 yes low
  523 89.8 1 yes high
/dev/cgroup # kill -9 440
/dev/cgroup # kill -9 523
[1]  Killed yes low > /dev/null
/dev/cgroup # taskset -c 1 yes high > /dev/null &
[3] 1216
[2]  Killed yes high > /dev/null
/dev/cgroup # echo $! > high/tasks
/dev/cgroup # taskset -c 1 yes low > /dev/null &
[4] 1404
/dev/cgroup # echo $! > low/tasks
/dev/cgroup # ps -C yes -opid,%cpu,psr,args

<table>
<thead>
<tr>
<th>PID</th>
<th>%CPU</th>
<th>PSR</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1216</td>
<td>83.3</td>
<td>1</td>
<td>yes high</td>
</tr>
<tr>
<td>1404</td>
<td>27.9</td>
<td>1</td>
<td>yes low</td>
</tr>
</tbody>
</table>
/dev/cgroup # killall -9 yes

[3]- Killed taskset -c 1 yes high > /dev/null


/dev/cgroup # echo 8096 > high/cpu.shares

/dev/cgroup # echo 8096 > low/cpu.shares

/dev/cgroup # taskset -c 1 yes low > /dev/null &

[1] 8187

/dev/cgroup # echo $! > low/tasks

/dev/cgroup # taskset -c 1 yes high > /dev/null &

[2] 8348

/dev/cgroup # echo $! > high/tasks

/dev/cgroup # ps -C yes -opid,%cpu,psr,args

  PID  %CPU  PSR  COMMAND

  8187  49.7   1 yes low
  8348  49.7   1 yes high
Subsystem: Cpuset
Subsystem: Cpuset

- Processor & Memory placement constraints for sets of tasks
- Cpuset defines a list of CPUs and memory nodes
  - CPUs include multiple processor cores as well as Hyper-Threads
  - memory nodes usually only one is available. NUMA (Non-Uniform Memory Access) platforms provide multiple memory nodes ...
- Subsystem is based on the (former) cputset Kernel implementation
  - cputset file system
  - Userspace tool: cset (SLERT10, SLES11, ...)

© B1 Systems GmbH 2006 – 2010 Chapter 5, Slide 39
~ # mount -tcgroup -ocpuset cpuset_example /dev/cgroup/

~ # cd /dev/cgroup/
/dev/cgroup # ls
cpuset.cpu_exclusive                       cpuset.memory_spread_slab
cpuset.cpus                                cpuset.mems
cpuset.mem_exclusive                       cpuset.sched_load_balance
cpuset.mem_hardwall                        cpuset.sched_relax_domain_level
cpuset.memory_migrate                      notify_on_release
cpuset.memory_pressure                     release_agent
cpuset.memory_pressure_enabled             tasks
cpuset.memory_spread_page
/dev/cgroup #
~ # taskset -p $$
pid 4235’s current affinity mask: 3
~ # taskset -c -p $$
pid 4235’s current affinity list: 0,1
~ # ps -o pid,psr,args

<table>
<thead>
<tr>
<th>PID</th>
<th>PSR</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>4235</td>
<td>1</td>
<td>-bash</td>
</tr>
<tr>
<td>4787</td>
<td>1</td>
<td>ps -o pid,psr,args</td>
</tr>
</tbody>
</table>
/dev/cgroup # mkdir cpuset1 cpuset2
/dev/cgroup # echo 0 > cpuset1,cpuset.cpus
/dev/cgroup # echo 0 > cpuset1,cpuset.mems
/dev/cgroup # echo 1 > cpuset2,cpuset.cpus
/dev/cgroup # echo 0 > cpuset2,cpuset.mems
/dev/cgroup # cd cpuset2; ps -o pid,psr
   PID  PSR
  4235   0
  4778   0
/dev/cgroup/cpuset2 # echo $$ > tasks
/dev/cgroup/cpuset2 # ps -o pid,psr
   PID  PSR
  4235   1
  4779   1
Cpuset

/dev/cgroup # rmdir cpuset2/
rmdir: failed to remove ‘cpuset2/’: Device or resource busy

/dev/cgroup # wc -l cpuset2/tasks
2 cpuset2/tasks
/dev/cgroup #

/dev/cgroup # for n in ‘cat cpuset2/tasks‘; do \
echo $n > tasks; done

-bash: echo: write error: No such process
/dev/cgroup # rmdir cpuset2/
/dev/cgroup #
Cpuset

```
/dev/cgroup # cat cpuset.cpus
0-3
/dev/cgroup # mkdir cpuset3
/dev/cgroup # echo 1,2,3 > cpuset3/cpuset.cpus
/dev/cgroup # cat cpuset3/cpuset.cpus
1-3
/dev/cgroup # echo 1-3 > cpuset3/cpuset.cpus
/dev/cgroup # cat cpuset3/cpuset.cpus
1-3
/dev/cgroup # echo 0,2-3 > cpuset3/cpuset.cpus
/dev/cgroup # cat cpuset3/cpuset.cpus
0,2-3
/dev/cgroup # echo "" > cpuset3/cpuset.cpus
/dev/cgroup # cat cpuset3/cpuset.cpus
```

```
/dev/cgroup # echo 3 > cpuset3/cpuset.cpus
/dev/cgroup # echo 1 > cpuset3/cpuset.cpu_exclusive
/dev/cgroup # echo 3 > cpuset2/cpuset.cpus
-bash: echo: write error: Invalid argument
/dev/cgroup # echo 0 > cpuset3/cpuset.cpu_exclusive
/dev/cgroup # echo 3 > cpuset2/cpuset.cpus
/dev/cgroup # mkdir cpuset3/sub3.1
/dev/cgroup # echo 0 > cpuset3/cpuset.cpu_exclusive
/dev/cgroup # echo 1 > cpuset3/sub3.1/cpuset.cpu_exclusive
-bash: echo: write error: Permission denied
/dev/cgroup # echo 1 > cpuset3/cpuset.cpu_exclusive
/dev/cgroup # echo 1 > cpuset3/sub3.1/cpuset.cpu_exclusive
/dev/cgroup #
/dev/cgroup # mkdir shield1 system
/dev/cgroup # echo 2-3 > shield1/cpuset.cpus
/dev/cgroup # echo 0 > shield1/cpuset.mems
/dev/cgroup # echo 0-1 > system/cpuset.cpus
/dev/cgroup # echo 0 > system/cpuset.mems
/dev/cgroup # echo 1 > shield1/cpuset.cpu_exclusive
/dev/cgroup # for n in ‘cat tasks‘; do \
    echo $n > system/tasks; done
-bash: echo: write error: Invalid argument
[...]
-bash: echo: write error: No such process
/dev/cgroup # wc -l tasks system/tasks shield1/tasks
32 tasks
126 system/tasks
0 shield1/tasks
158 total
Cpuset

```bash
/dev/cgroup # ps -p 'cat tasks'

PID  TTY  STAT  TIME    COMMAND
  3   ?    S<   0:00   [migration/0]
  4   ?    S<   0:00   [ksoftirqd/0]
  5   ?    S<   0:01   [migration/1]
  6   ?    S<   0:00   [ksoftirqd/1]
[...]
  96  ?    S<   0:00   [ata/0]
  97  ?    S<   0:02   [ata/1]
  98  ?    S<   0:00   [ata/2]
  99  ?    S<   0:00   [ata/3]

/dev/cgroup # cat /proc/self/cgroup
1:cpuset:/system

/dev/cgroup # echo $$ > shield1/tasks
/dev/cgroup # cat /proc/self/cgroup
1:cpuset:/shield1
```
Subsystem: Memory
~ # mount -t cgroup -o memory memory_example /dev/cgroup
~ # cd /dev/cgroup/; ls memory.*
memory.failcnt  memory.max_usage_in_bytes
memory.force_empty memory.stat
memory.limit_in_bytes memory.usage_in_bytes

[...]
/dev/cgroup # mkdir mem1; cd mem1/
/dev/cgroup/mem1 # echo $$ > tasks
/dev/cgroup/mem1 # cat memory.usage_in_bytes
208896
/dev/cgroup/mem1 # cat memory.limit_in_bytes
9223372036854775807
/dev/cgroup/mem1 # echo 512M > memory.limit_in_bytes
/dev/cgroup/mem1 # cat memory.limit_in_bytes
536870912
Libcgroup
What Is Libcgroup?

Using the plain cgroup file systems has following disadvantages:

- it is not persistent, after a reboot everything is gone
- requires to write init scripts to set up cgroups (maintenance?)
- not all users are familiar to the special behavior of the cgroup file system
- tasks might leak and run in root cgroup if parent process is not also in a non-cgroup
- tasks do not get automatically reassigned to the "right" cgroup
Libcgroup tries to fill the gap of the missing user-space part. It consists of:

- shared library with a generic cgroup userspace API: libcgroup.so
- PAM Module: pam_cgroup.so
- Command Line tools: cgexec, cgclassify, ...
- Daemon: cgrulesengd
Libcgroup command line tools

- `cgconfigparser` - Used for parsing a configuration file and maintaining persistence across reboots.
- `cgclear` - Destroy all control group hierarchies
- `cgexec` - Start a process in a cgroup
- `cgred` - Automatic classification daemon originally based on user classification. Now enhanced for process based classification as well.
- `cgset / cgget` - List cgroup values
- `lscggroup` - List all cgroups
- `cgsnapshot` - (Beta) Generate configurations from current setup

Some more, check the libcgroup1 package on your system.
The cgroups configuration parser of cgconfig.cfg is available in multiple variants:

- (developers) libcgroup API:
  ```c
  int cgroup_config_load_config(const char *pathname)
  ```
- `/usr/sbin/cgconfigparser`
- `/etc/init.d/cgconfig`
  - reads `/etc/cgconfig.conf`
  - creates by default a sysdefault cgroup

```
~ # wc -l /etc/cgconfig.conf
22 /etc/cgconfig.conf
~ # /etc/init.d/cgconfig start
Starting service cgconfig done
~ # ls /cgroup/
cpu.shares notify_on_release release_agent release_agent tasks
```

© B1 Systems GmbH 2006 – 2010
libcgroup configuration file to define control groups ...

```plaintext
group professors {
    perm {
        task {
            uid = tux;
            gid = professors;
        }
        admin {
            uid = root;
            gid = root;
        }
    }
    cpu {
        cpu.shares = 500;
    }
}
```
... and mount points of the cgroup file system:

[...]
mount {
    cpu = /cgroup;
    cpuacct = /cgroup;
}

cgrules.conf is the second libcgroup configuration file and holds rules about which tasks should get assigned to which cgroup.

```
~ # tail -n3 /etc/cgrules.conf
#<user> <subsystems> <destination>
tux cpu professor/tux/
@professors cpu,cpuacct professor/
```
cgexec is a command line tool to execute and assign tasks into a specific control group:

\[\text{cgexec } [-g \text{ <list of controllers>:<relative path to cgroup>}] \text{ command [arguments]}\]

- \text{cgexec -g *:professors ls}
- \text{cgexec -g cpu,memory:professors ls -lisa}
- \text{cgexec -g cpu,memory:professors -g cpuset:shield1 ls -ltr}

If parameter \(-g\) is not supplied the tools assigns the task to the first matching rule from \(\text{/etc/cgrules.conf}\).
cgclassify assigns already running tasks based on /etc/cgrules.conf to a matching cgroup.

- cgclassify <list of pids>
- cgclassify 3323 4210
As an alternative to manually distributing tasks, tasks can automatically be distributed based on `/etc/cgrules.conf` with the Cgroups Rules Engine Daemon.

```
~ # /etc/init.d/cgred start
Starting CGroup Rules Engine Daemon Log file is: /var/log/cgred
Starting in daemon mode.
Opened log file: /var/log/cgred
~ # tail -f /var/log/cgred
GID Event:
  PID = 7019, tGID = 7019, rGID = 100, eGID = 100
  Attempting to change cgroup for PID: 7019, UID: 1000, GID: 100
[...]
```
Subsystem: CPU Accounting Controller
CPU Accounting Controller accounts the CPU usage:

- of tasks in a cgroup
- and of its child cgroups (if available)
~ # mount -tcgroup -ocpuacct cpuacct_example /dev/cgroup
~ # cd /dev/cgroup/; ls
 cpuacct.usage notify_on_release release_agent tasks 
/dev/cgroup # mkdir cpuacct1; cd cpuacct1/; ls
 cpuacct.usage notify_on_release tasks 
/dev/cgroup/cpuacct1 # mount
[
...
]
 cpuacct_example on /dev/cgroup type cgroup (rw,cpuacct) 
/dev/cgroup/cpuacct1 # cat cpuacct.usage
 0
 /dev/cgroup/cpuacct1 # echo $$ > tasks
 /dev/cgroup/cpuacct1 # cat cpuacct.usage
 5477290
 /dev/cgroup/cpuacct1 # yes > /dev/null &
 /dev/cgroup/cpuacct1 # cat cpuacct.usage
 2114152710
Subsystem: Devices
The *Devices* subsystem is also called: *Device Whitelist Controller*

```bash
~ # mount -t cgroup -o devices devices_example /dev/cgroup
~ # cd /dev/cgroup/; ls -l devices.*
devices.allow
devices.deny
devices.list
/dev/cgroup # cat devices.list
a *:* rwm
```

```bash
/dev/cgroup # mkdir devices1; cd devices1/
/dev/cgroup/devices1 # ls -l devices.*
devices.allow
devices.deny
devices.list
/dev/cgroup/devices1 # cat devices.list
a *:* rwm
```
A whitelist entry consists of four fields:

- **type** stands for the entry type:
  - a applies to all types and major&minor numbers
  - c character device
  - b block device

- **major number** major number as integer, or * for all

- **minor number** minor number as integer, or * for all

- **access** access modes:
  - r read
  - w write
  - m mknod
Allow everything:

```bash
# echo "a *:* rwm" > devices.allow
```

Deny everything:

```bash
# echo "a *:* rwm" > devices.deny
```

Allow read-only access to SCSI disk devices (0-15):

```bash
# echo "b 8:* r" > devices.deny
```

(Linux allocated devices:
/usr/src/linux/Documentation/devices.txt)
Subsystem: Freezer
~ # mount -t cgroup -o freezer freezer_example /dev/cgroup
~ # cd /dev/cgroup/
/dev/cgroup # mkdir freezer1
/dev/cgroup # ls
freezer1 notify_on_release release_agent tasks
/dev/cgroup # cd freezer1/
/dev/cgroup/freezer1 # ls
freezer.state notify_on_release tasks
/dev/cgroup/freezer1 # cat freezer.state
THAWED
/dev/cgroup/freezer1 #
Subsystem Namespace

Subsystem *Namespace*
~ # mkdir /dev/cgroup
~ # mount -tcgroup -ons namespace_example /dev/cgroup
~ # cd /dev/cgroup/
/dev/cgroup # ls
notify_on_release release_agent tasks
/dev/cgroup # /root/newns
/dev/cgroup # ls
3434 notify_on_release release_agent tasks
/dev/cgroup # echo $$
3434
/dev/cgroup # /root/newns
/dev/cgroup # find -type d
.
./3434
./3434/3446