Linux containers

You are not expected to understand this

Bruno Barcarol Guimarães

bbguimaraes.com

2016-10-06

overview

1. overview
   - introduction
   - pre-history
   - history
2. implementation
   - capabilities
   - seccomp
   - cgroups
   - namespaces
3. references

overview/introduction

You are not expected to understand this.

overview/introduction

You are not expected to understand this.

Dennis Ritchie

overview/introduction

```
/*
 * Switch to stack of the new process and set up
 * his segmentation registers.
 */
retu(rp->p_addr);
sureg();
/*
 * If the new process paused because it was
 * swapped out, set the stack level to the last call
 * to savu(u_ssav).  This means that the return
 * which is executed immediately after the call to aretu
 * actually returns from the last routine which did
 * the savu.
 *
 * You are not expected to understand this.
 */
if(rp->p_flag&SSWAP) {
    rp->p_flag =& ~SSWAP;
    aretu(u.u_ssav);
}
/*
 * The value returned here has many subtle implications.
 * See the newproc comments.
 */
return(1);
```

Dennis Ritchie, Sixth Edition Unix (1975) [1]

overview/introduction

> *"You are not expected to understand this" was intended as a remark in the spirit of "This won't be on the exam," rather than as an impudent challenge.*

Dennis Ritchie [2]

The real problem is that we didn't understand what was going on either.

The savu/retu mechanism for doing process exchange was fundamentally broken because it depended on switching to a previous stack frame and executing function return code in a different procedure from the one that saved the earlier state. This worked on the PDP-11 because its compiler always used the same context-save mechanism; with the Interdata compiler, the procedure return code differed depending on which registers were saved.

So, for Steve Johnson and me, trying to move the kernel for the first time to a new machine, this code was indeed on the exam. It took about a week of agonizing before we finally convinced each other that the mechanism was wrong and no fiddling with the compiler was useful. We redid the coroutine control-passing primitives altogether, and this code section, and the comment, passed into history.

Dennis Ritchie

overview/pre-history

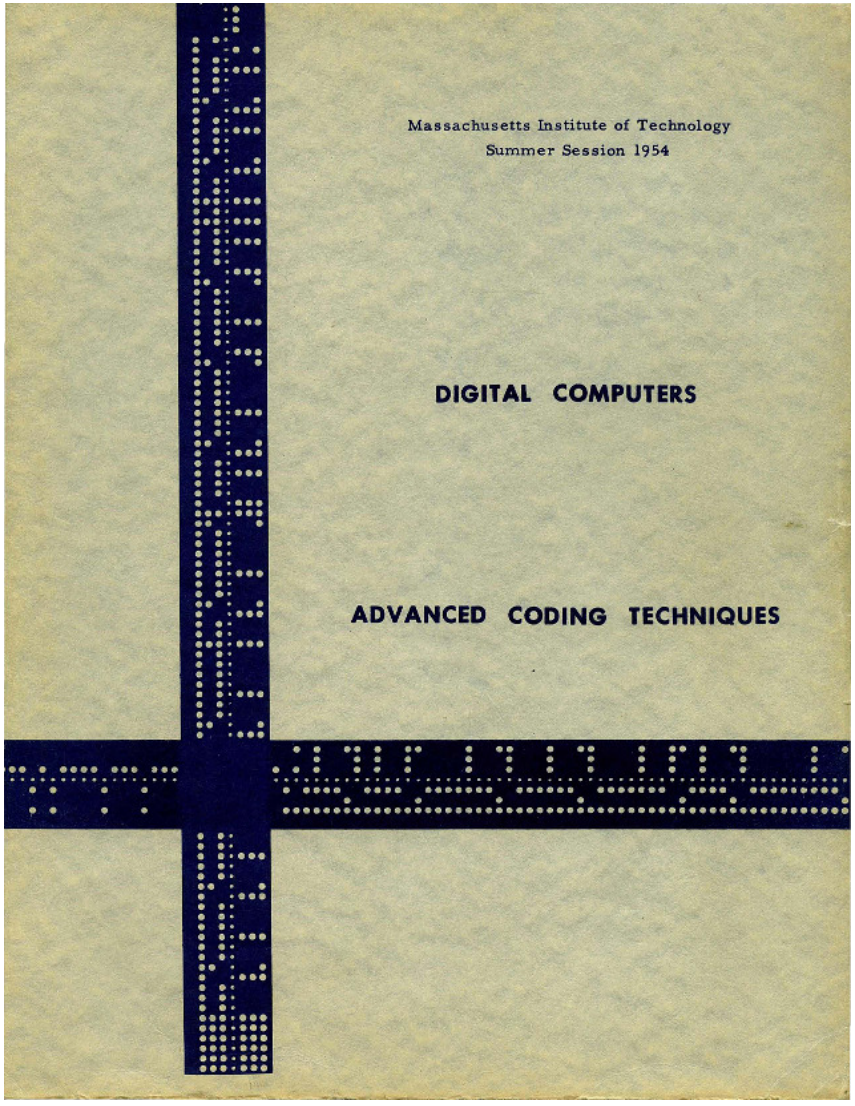
> *You have probably heard about Docker and Containers, but FreeBSD have had jails since I wrote them in 1998.*

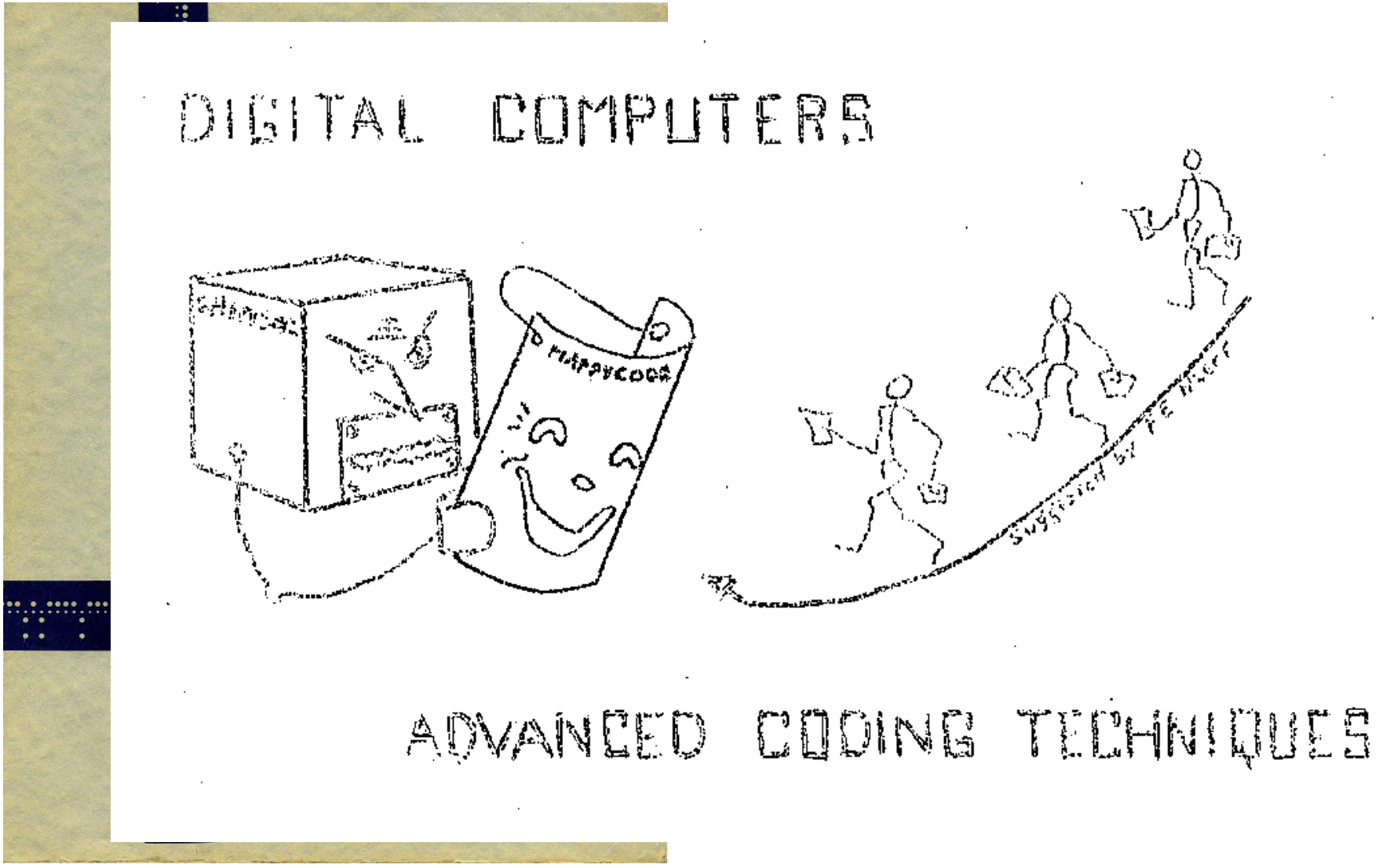
Poul-Henning Kamp [3]

overview/pre-history/time-sharing

- First appearance of the term by Josh Backus in a MIT Summer Session (1954)
- Project to implement a time-sharing system started by John McCarthy at MIT (1959)
- Compatible Time-Sharing System, CTSS (1961)

overview/pre-history/time-sharing




DIGITAL COMPUTERS
ADVANCED CODING TECHNIQUES

> *Mr. P. F. Williams said that his firm is trying to decide on a computer to use. They want something intermediate between an IBM 650 and 704. The 704 seems too large: they would not be able to keep it busy. Josh Backus said that by time sharing, a big computer could be used as several small ones; [...]*

Computer Advanced Coding Techniques, MIT Summer Session [4]

overview/pre-history/time-sharing

> *By allowing a large number of users to interact concurrently with a single computer, time-sharing dramatically lowered the cost of providing computing capability. [...] While any single user would make inefficient use of a computer, a large group of users together would not.*

Wikipedia [5]

overview/history/chroot

- `chroot(2)`
- Changes the root directory of the calling process
- Compartmentalize the filesystem
- Didn't provide any other isolation mechanisms
- Very trivial to "escape"
- The goal was not security
- Actually, no one really knows what the goal was

[CHROOT]

Dr. Marshall Kirk Mckusick, private communication: "According to the SCCS logs, the chroot call was added by Bill Joy on March 18, 1982 approximately 1.5 years before 4.2BSD was released. That was well before we had ftp servers of any sort (ftp did not show up in the source tree until January 1983). My best guess as to its purpose was to allow Bill to chroot into the /4.2BSD build directory and build a system using only the files, include files, etc contained in that tree. That was the only use of chroot that I remember from the early days."

Poul-Henning Kamp, Robert Watson [6]

overview/history/jails

- `jail(2)`
- Added to FreeBSD by Poul-Henning Kamp (1999)
- "Confining the omnipotent root"
- Created to address the problem of needing many machines, each almost idle, to have different versions of apache, mysql, perl etc.
- "Trivial extension" to `chroot(2)` to become "light-weight virtual 'machines'"

overview/history/jails

> *The FreeBSD "Jail" facility provides the ability to partition the operating system environment, while maintaining the simplicity of the UNIX "root" model. In Jail, users with privilege find that the scope of their requests is limited to the jail, allowing system administrators to delegate management capabilities for each virtual machine environment.*

Poul-Henning Kamp, Robert Watson [6]

overview/history/jails

- "one-way mirror effect"
  - unjailed processes can see jailed processes, send signals, attach debuggers, etc. (still subject to Unix access controls)
  - jailed processes can't see or interact with processes outside, either unjailed or in other jails

If somebody breaks into your computer, you face a nasty set of problems. For instance if you open a terminal connection, you may be talking to their special version of the sshd(8) daemon, and who knows what that does?

If you are running on perfectly virtualized servers, such as VMware, Xen or similar, you are as much subject to these problems as you are with dedicated hardware.

But if you put your outward facing services in a jail, you can comfortably log in using the unadultered sshd(8) in the unjailed part of the system, and see what the attackers are up to, while they cannot see you.

Poul-Henning Kamp [7]

overview/history/linux

- Virtuozzo
  - SWsoft, now Parallels (2000)
  - originally proprietary
  - switched to open-core (OpenVZ, GPL) model (2005) [8]
- Virtual private servers and security contexts
  - Jacques Gélinas (2001) [9]
- Linux-VServer
  - Herbert Pötzl (2002) [10]
- all require (to this day) a patched kernel

overview/history/zones

- `zone(2)`
- Solaris 10 (2004)
- Proprietary, open-sourced (CDDL) as OpenSolaris (2005)
- Integrated at the operating system level
- Server consolidation
  - namespace isolation
  - security isolation
  - quality of service guarantees
  - account for resource utilization

overview/history/zones

> *Managing zones is not complicated. Figure 2 shows how to create a simple, non-networked zone called lisa with a file system hierarchy rooted at /aux0/lisa, install the zone, and boot it. Booting a zone causes the init daemon for the zone to be launched. At that point, the standard system services such as cron, sendmail, and inetd are launched.*

Daniel Price, Andrew Tucker [11]

overview/history/zones

- Platform administrator vs. application administrator systems
- Installation
  - package-manager aware
  - packages can be managed from both the global and local zone
  - reuses (copy or hard-link) files from the global zone
  - sparse-root zones: `lofs` `/usr`, `/lib`, `/sbin`, `/platform`
- Branded zones
  - syscall translation layer
  - allows executing non-native binaries
  - `solaris8`, `solaris9`, `lx`

implementation/capabilities

- Attempted standardization with POSIX.1e (abandoned)
- Implemented differently in each operating system
  - Linux's capabilities (1999)
  - Solaris's privileges
  - FreeBSD's capsicum

implementation/capabilities

> *Starting with kernel 2.2, Linux divides the privileges traditionally associated with superuser into distinct units, known as capabilities, which can be independently enabled and disabled.*

`capabilities(7)`

implementation/capabilities

- Per-thread attribute
- Set of 64 (originally 32) bits
- Inherited from the parent
- Can be acquired by executing a file with associated capabilities
- Examples: `CAP_CHOWN`, `CAP_KILL`, `CAP_MKNOD`, `CAP_NET_RAW`, `CAP_SETPCAP`, `CAP_SYS_ADMIN`, `CAP_SYS_TRACE`

implementation/seccomp

- `prctl(PR_SET_SECCOMP, 1)`
  - linux 2.6.23 (2007)
  - `read(2)`, `write(2)`, `_exit(2)`, `sigreturn(2)`
  - `SIGKILL`
- `seccomp(2)`
  - linux 3.17 (2014)
  - `SECCOMP_MODE_STRICT`
  - `SECCOMP_MODE_FILTER`

implementation/seccomp

```
/* <linux/seccomp.h> */
/* input */
struct seccomp_data {
    int   nr;
    __u32 arch;
    __u64 instruction_pointer;
    __u64 args[6];
};
/* output */
#define SECCOMP_RET_KILL   /* ... */
#define SECCOMP_RET_TRAP   /* ... */
#define SECCOMP_RET_ERRNO  /* ... */
#define SECCOMP_RET_TRACE  /* ... */
#define SECCOMP_RET_ALLOW  /* ... */
```

implementation/cgroups

- linux 2.6.24 (2007)
- fine-grained resource partitioning among competing (groups of) processes
  - resource limiting/reservation
  - accounting/auditing
- expanded the concept of cpusets that was already present in the kernel (2004)
- no new system calls, all the management is performed through a virtual filesystem
- process grouping based on hierarchies (trees)

implementation/cgroups

- v2 in linux 4.5 (2016-03-14)
- unified hierarchy
- process granularity
- processes only on leaf nodes
- single arbiter process
- general cleanup

implementation/cgroups

```
$ column -t /proc/cgroups
#subsys_name  hierarchy  num_cgroups  enabled
cpuset        10         10           1
cpu           5          95           1
cpuacct       5          95           1
blkio         3          100          1
memory        8          365          1
devices       9          97           1
freezer       11         10           1
net_cls       4          10           1
perf_event    7          10           1
net_prio      4          10           1
hugetlb       2          9            1
pids          6          102          1
```

implementation/cgroups

```
$ awk '/cgroup/{print$2,$3}' /proc/mounts
/sys/fs/cgroup tmpfs
/sys/fs/cgroup/systemd cgroup
/sys/fs/cgroup/blkio cgroup
/sys/fs/cgroup/net_cls,net_prio cgroup
/sys/fs/cgroup/cpu,cpuacct cgroup
/sys/fs/cgroup/pids cgroup
/sys/fs/cgroup/perf_event cgroup
/sys/fs/cgroup/memory cgroup
/sys/fs/cgroup/devices cgroup
/sys/fs/cgroup/freezer cgroup
/sys/fs/cgroup/cpuset cgroup
```

implementation/cgroups

```
$ sed -n '/name=systemd/s/,\| /\n/gp' /proc/mounts
cgroup
/sys/fs/cgroup/systemd
cgroup
rw
nosuid
nodev
noexec
relatime
xattr
release_agent=/usr/lib/systemd/systemd-cgroups-agent
name=systemd
0
0
```

implementation/cgroups

```
$ ls -l /sys/fs/cgroup/systemd
total 0
-rw-r--r--.  1 root root 0 Sep  2 14:49 cgroup.clone_children
-rw-r--r--.  1 root root 0 Sep  2 14:49 cgroup.procs
-r--r--r--.  1 root root 0 Sep  2 14:49 cgroup.sane_behavior
drwxr-xr-x.  2 root root 0 Sep  2 11:35 init.scope
drwxr-xr-x.  2 root root 0 Sep  2 12:13 machine.slice
-rw-r--r--.  1 root root 0 Sep  2 14:49 notify_on_release
-rw-r--r--.  1 root root 0 Sep  2 14:49 release_agent
drwxr-xr-x. 92 root root 0 Sep  2 13:05 system.slice
-rw-r--r--.  1 root root 0 Sep  2 14:49 tasks
drwxr-xr-x.  3 root root 0 Sep  2 11:35 user.slice
```

implementation/cgroups

```
$ awk '$3=="cgroup"&&!/systemd/{print$2,$4}' /proc/mounts
/sys/fs/cgroup/cpuset rw,nosuid,nodev,noexec,relatime,cpuset
/sys/fs/cgroup/memory rw,nosuid,nodev,noexec,relatime,memory
/sys/fs/cgroup/net_cls,net_prio rw,nosuid,nodev,noexec,relatime,net_cls,net_prio
/sys/fs/cgroup/freezer rw,nosuid,nodev,noexec,relatime,freezer
/sys/fs/cgroup/blkio rw,nosuid,nodev,noexec,relatime,blkio
/sys/fs/cgroup/cpu,cpuacct rw,nosuid,nodev,noexec,relatime,cpu,cpuacct
/sys/fs/cgroup/hugetlb rw,nosuid,nodev,noexec,relatime,hugetlb
/sys/fs/cgroup/perf_event rw,nosuid,nodev,noexec,relatime,perf_event
/sys/fs/cgroup/devices rw,nosuid,nodev,noexec,relatime,devices
/sys/fs/cgroup/pids rw,nosuid,nodev,noexec,relatime,pids
```

implementation/cgroups/cpuset

```
$ cat /sys/fs/cgroup/cpuset/cpuset.cpus
0-3
$ echo 0-1 > /sys/fs/cgroup/cpuset/cpuset.cpus
$ cat /sys/fs/cgroup/cpuset/cpuset.mems
0
$ echo 3-5 > /sys/fs/cgroup/cpuset/cpuset.mems
```

implementation/cgroups/cpuset

```
$ ls /sys/fs/cgroup/cpuset | grep 'cpuset\.'
cpuset.cpu_exclusive
cpuset.cpus
cpuset.effective_cpus
cpuset.effective_mems
cpuset.mem_exclusive
cpuset.mem_hardwall
cpuset.memory_migrate
cpuset.memory_pressure
cpuset.memory_pressure_enabled
cpuset.memory_spread_page
cpuset.memory_spread_slab
cpuset.mems
cpuset.sched_load_balance
cpuset.sched_relax_domain_level
```

implementation/cgroups/cpu

```
$ cat /sys/fs/cgroup/cpu/cpu.shares
1024
$ echo 2048 > /sys/fs/cgroup/cpu/cpu.shares
$ ls /sys/fs/cgroup/cpu | grep 'cpu\.cfs_'
cpu.cfs_period_us
cpu.cfs_quota_us
$ awk '{print$1}' /sys/fs/cgroup/cpu/cpu.stat
nr_periods
nr_throttled
throttled_time
```

implementation/cgroups/cpuacct

```
$ ls /sys/fs/cgroup/cpuacct | grep 'cpuacct\.'
cpuacct.stat
cpuacct.usage
cpuacct.usage_percpu
cpuacct.usage_percpu_sys
cpuacct.usage_percpu_user
cpuacct.usage_sys
cpuacct.usage_user
$ cat /sys/fs/cgroup/cpuacct/cpuacct.stat
user 50513
system 5274
$ cat /sys/fs/cgroup/cpuacct/cpuacct.usage
567909305094
$ cat /sys/fs/cgroup/cpuacct/cpuacct.usage_percpu
119426195198 182347519425 112831274244 155276137008 0 0 0 0
```

implementation/cgroups/blkio

```
$ ls /sys/fs/cgroup/blkio \
    | awk '/blkio\./&&!/recursive|throttle/'
blkio.io_merged
blkio.io_queued
blkio.io_service_bytes
blkio.io_serviced
blkio.io_service_time
blkio.io_wait_time
blkio.leaf_weight
blkio.leaf_weight_device
blkio.reset_stats
blkio.sectors
blkio.time
blkio.weight
blkio.weight_device
```

implementation/cgroups/blkio

```
$ ls /sys/fs/cgroup/blkio | grep 'blkio\.throttle'
blkio.throttle.io_service_bytes
blkio.throttle.io_serviced
blkio.throttle.read_bps_device
blkio.throttle.read_iops_device
blkio.throttle.write_bps_device
blkio.throttle.write_iops_device
```

implementation/cgroups/perf_event

```
$ ls /sys/fs/cgroup/perf_event | grep -c 'perf_event\.'
0
```

implementation/cgroups/memory

```
$ cat /sys/fs/cgroup/memory/memory.usage_in_bytes
3880935424
$ cat /sys/fs/cgroup/memory/memory.limit_in_bytes
9223372036854771712
$ echo 1 > /sys/fs/cgroup/memory/memory.limit_in_bytes
$ echo 1 > /sys/fs/cgroup/memory/memory.soft_limit_in_bytes
```

implementation/cgroups/memory

```
$ awk '!/^total_/{print$1}' /sys/fs/cgroup/memory/memory.stat
cache                                   inactive_anon
rss                                     active_anon
rss_huge                                inactive_file
mapped_file                             active_file
dirty                                   unevictable
writeback                               hierarchical_memory_limit
swap                                    hierarchical_memsw_limit
pgpgin                                  recent_rotated_anon
pgpgout                                 recent_rotated_file
pgfault                                 recent_scanned_anon
pgmajfault                              recent_scanned_file
```

implementation/cgroups/devices

```
$ # a - all,  c - char,  b - block
$ # r - read, w - write, m - mknod
$ cat /sys/fs/cgroup/devices/devices.list
a *:* rwm
$ echo a > /sys/fs/cgroup/devices/devices.deny
$ # major:minor, 1:3 -> /dev/null
$ echo 'c 1:3 rwm' > /sys/fs/cgroup/devices/devices.allow
```

implementation/cgroups/freezer

```
$ ls /sys/fs/cgroup/freezer | grep -c 'freezer\.'
0
$ mkdir /sys/fs/cgroup/freezer/child
$ cat /sys/fs/cgroup/child/freezer.state
THAWED
$ echo FROZEN > /sys/fs/cgroup/child/freezer.state
$ echo THAWED > /sys/fs/cgroup/child/freezer.state
```

implementation/cgroups/net_cls

```
$ cat /sys/fs/cgroup/net_cls/net_cls.classid
0
$ maj=0001; min=0001
$ echo 0x$maj$min > /sys/fs/cgroup/net_cls/net_cls.classid
```

implementation/cgroups/net_prio

```
$ cat /sys/fs/cgroup/net_prio/net_prio.prioidx
1
$ echo 2 > /sys/fs/cgroup/net_prio/net_prio.prioidx
-bash: echo: write error: Invalid argument
$ cat /sys/fs/cgroup/net_prio/net_prio.ifpriomap
lo 0
enp0s25 0
$ echo enp0s25 5 > /sys/fs/cgroup/net_prio/net_prio.ifpriomap
```

implementation/cgroups/hugetlb

```
$ ls /sys/fs/cgroup/hugetlb | grep 'hugetlb\.'
hugetlb.1GB.failcnt
hugetlb.1GB.limit_in_bytes
hugetlb.1GB.max_usage_in_bytes
hugetlb.1GB.usage_in_bytes
hugetlb.2MB.failcnt
hugetlb.2MB.limit_in_bytes
hugetlb.2MB.max_usage_in_bytes
hugetlb.2MB.usage_in_bytes
```

implementation/cgroups/pid

```
$ ls /sys/fs/cgroup/pids | grep -c 'pids\.'
0
$ mkdir /sys/fs/cgroup/pids/child
$ cat /sys/fs/cgroup/child/pids/pids.current
0
$ cat /sys/fs/cgroup/pids/pids.max
max
$ echo 2 > /sys/fs/cgroup/pids/pids.max
```

implementation/namespaces

- Wraps a global resource, processes within the namespace appear to have their own isolated instance.
- Operations that would affect the global resource are restricted to the namespace of the process and don't affect other namespaces.

implementation/namespaces

- `unshare(2)`
- `clone(2)`
- `setns(2)`

implementation/namespaces

- mnt (`CLONE_NEWNS`)
- uts (`CLONE_NEWUTS`)
- ipc (`CLONE_NEWIPC`)
- pid (`CLONE_NEWPID`)
- net (`CLONE_NEWNET`)
- user (`CLONE_NEWUSER`)
- cgroup (`CLONE_NEWCGROUP`)

implementation/namespaces

```
# ls -l /proc/1/ns
total 0
lrwxrwxrwx. 1 root root 0 Sep  2 19:56 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx. 1 root root 0 Sep  2 19:56 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx. 1 root root 0 Sep  2 19:56 mnt -> 'mnt:[4026531840]'
lrwxrwxrwx. 1 root root 0 Sep  2 19:56 net -> 'net:[4026531969]'
lrwxrwxrwx. 1 root root 0 Sep  2 19:56 pid -> 'pid:[4026531836]'
lrwxrwxrwx. 1 root root 0 Sep  2 19:56 user -> 'user:[4026531837]'
lrwxrwxrwx. 1 root root 0 Sep  2 19:56 uts -> 'uts:[4026531838]'
```

implementation/namespaces/mount

- `CLONE_NEWNS`
- linux 2.4.19 (2002)
- `mount(2)`/`umount(2)`
- different processes have different visions of the file system
- "`chroot(2)` on steroids"
- masking of `/proc`, `/sys`, etc.
- sharing of mount points

implementation/namespaces/uts

- `CLONE_NEWUTS`
- linux 2.6.19 (2006)
- `uname(2)`
- `sethotname(2)`
- `setdomainname(2)`
- Unix Timesharing System

implementation/namespaces/ipc

- `CLONE_NEWIPC`
- linux 2.6.19 (2006) / linux 2.6.30 (2009)
- a.k.a. "the namespace no one knows what it's for"
- `svipc(7)`/`mq_overview(7)`
- isolation for the objects that are not identified by files
- System V ipc and POSIX message queues have been mostly superseded

implementation/namespaces/pid

- `CLONE_NEWPID`
- linux 2.6.24 (2008)
- each pid inside a namespace is mapped to a pid outside
- can only be created by `clone(2)`ing, the child process becomes pid 1 in the namespace
- when pid 1 exits, all processes on the namespace are killed
- processes don't see processes of other namespaces and can't use system calls such as `kill(2)`, `ptrace(2)`, etc
- `procfs` only shows information related to processes on the namespace of the process that mounted it
- can be nested: a parent namespace can see and affect processes on child namespaces; processes on a child namespace don't see and can't affect processes on the parents

implementation/namespaces/net

- `CLONE_NEWNET`
- linux 2.6.24 (2008)
- each namespace has its own network devices (including `lo`), ip addresses, routing tables, `/proc/net`, `/sys/class/net`, ports, etc.
- communication with the host or other namespaces can be implemented using `veth` pairs and bridge interfaces
- `INADDR_ANY` (a.k.a. `0.0.0.0`) means "any address on the current namespace"
- is also used for accounting (`iptables(8)`, `netlink(7)`, etc.), as interfaces are per-namespace

implementation/namespaces/user

- `CLONE_NEWUSER`
- linux 2.6.23 (2007)
- uid and gid mapping
- can be nested
- the process receives the full set of capabilities upon entering a user namespace, but has no capabilities on parent user namespaces
- since linux 3.8 (2013), unprivileged users can create a user namespace
  - other namespaces can be created inside the user namespace
  - `uid 0` inside the namespace

implementation/namespaces/user

- `CLONE_NEWUSER`
- linux 2.6.23 (2007)
- uid and gid mapping
- can be nested
- the process receives the full set of capabilities upon entering a user namespace, but has no capabilities on parent user namespaces
- since linux 3.8 (2013), unprivileged users can create a user namespace
  - other namespaces can be created inside the user namespace
  - `uid 0` inside the namespace
  - ö

implementation/namespaces/cgroup

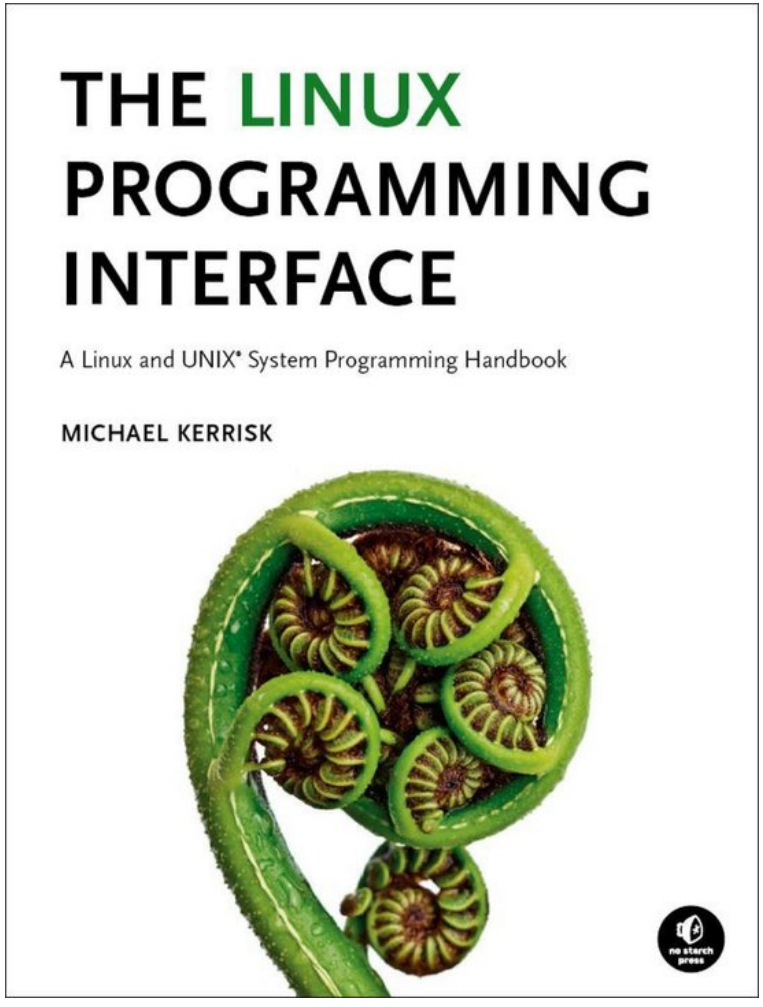
- `CLONE_NEWCGROUP`
- linux 4.6 (2016)
- the process' cgroup becomes the root of the cgroup hierarchy inside the namespace
- hides the cgroup hierarchy
- prevents modification of cgroup configuration on parent cgroups (if file permission allowed it)

> *These two papers are important because they don't just capture what was done, but why it was done. This is really, really, really important. And if you are working on an important technology, please do this. Because when we don't have it, we have javascript.*

Bryan Cantrill [12]

references

[13]



[14]

references I

- The Linux Programming Interface
- Docker drivers

references I

- /usr/sys/ken/slp.c - Sixth Edition Unix source code
- Odd Comments and Strange Doings in Unix
- Varnish - How our website works
- MIT - Computer Advanced Coding Techniques
- Wikipedia - Time-sharing
- Jails: Confining the omnipotent root
- Jails: High value but shitty Virtualization
- OpenVZ
- Virtual private servers and security contexts
- Linux-VServer

references II


Daniel Price, Andrew Tucker - Solaris Zones: Operating System Support for Consolidating Commercial Workloads


Bryan Cantrill - Jails and Solaris Zones


https://lwn.net/subscribe/Info


http://www.man7.org/tlpi/