Operating system level virtualisation

COSC349—Cloud Computing Architecture

David Eyers

Learning objectives

- Enumerate multiple motivations for **resource isolation**
- Can define **OS-level virtualisation**
- Explain a benefit & a downside of OS-level virtualisation
- Appreciate that OS-level virtualisation is an old idea
- Can describe the role of **Linux namespaces and cgroups** in effecting Linux-based OS-level virtualisation

Motivations for isolation of resources

- Typical motivation presented so far has been **security**
  - Maintain confidentiality & integrity of separate users' data
- Isolation can also be to support software **manageability**
  - Applications that need specific, conflicting support software versions
    - Runtime environments may allow local installation, e.g., Python 'virtualenv's
  - Want to be able to install and cleanly remove sets of software
    - Linux distribution package managers can provide this support
- Also to support **testing** within software development
  - Allow test environments to be created and cleanly destroyed, rapidly

Computing has many types of isolation

- **Application**-level (i.e., in-application) isolation
- **Threads**—share memory within one process
- OS **processes**—each has its own address space
- **Userspaces**—i.e., everything above the OS kernel
- **Virtual machines**—full or paravirtualised

- Today's lecture focus: **isolating separate userspaces**
  - Also known as OS-level virtualisation—leading toward Docker

Cheaper isolation if OS kernel is secure

- Trusting OS kernel security allows for **cheap isolation**
  - (i.e., cheaper than needing to run VMs containing OS kernels)
- We have talked about userspace / kernel separation
  - Also consider user / user separation
  - Multi-user OSs assume user processes are successfully isolated
- Android embodies this, by allocating user IDs for apps
  - Thus each application's processes are (assumed) isolated

“Old school” chroot jails

- Unix servers have to handle **users that may be malign**
  - Common historical example was running public FTP servers
  - Anonymous users could log into those servers
  - FTP as a protocol allows quite a lot of power over the server
  - Needed to cut down what anonymous users could do
- Solution: change the perceived **root of the filesystem**
  - *i.e.*, a ‘chroot jail’—usefully changes available executables
  - Unix accesses binaries from `/bin`, libraries from `/lib`, *etc.*
  - Changing the meaning of / mitigates many vulnerabilities

BSD Jails—OS-level virtualisation since Y2K

- BSD Jails take resource partitioning beyond the filesystem
  - Isolate process IDs, root user, network, device access
  - Also use a chroot jail to effect filesystem isolation
- Can help avoiding privilege escalation
  - Successful break in to server can't scan filesystem for vulnerabilities, *e.g.*, reading `/etc/shadow` and trying to crack weak passwords
- Many operations are blocked within BSD Jaills, *e.g.*:
  - loading kernel modules, changing network interfaces, mounting and unmounting filesystems, *etc*.

Linux-vserver—Linux follows BSD in 2001

- Its isolation groups named **virtual private servers** (VPS)
  - Organisations used to run web server in “colo” data centres
    - Data centres offer reliable power, internet connectivity, *etc*.
    - You co-located your servers with others’ in the data centre
  - Want to aggregate these web servers, but isolate resources
- Starting a VPS involves starting another `init` process
  - `init` has process ID 1 and is the **parent of all Linux processes**
  - Isolation rather than virtualisation of storage and NICs
  - e.g., map VPS’ files into subtrees of single filesystem

Solaris Zones—2004

- Solaris was Sun’s Unix variant. Version 10 introduced:
  - Solaris ‘Zones’—i.e., separate userspaces over one kernel
  - Solaris ZFS—a copy-on-write filesystem with zones support
  - DTrace—in-kernel debugging (ported to BSDs including macOS)
- Solaris was, at least historically, more secure than Linux
  - … it was also much more expensive than Linux
  - Sun later open-sourced Solaris… then the company imploded
  - Oracle still support & sell Solaris; also many open-source variants

Solaris ZFS filesystem

- ZFS was one of the mainstream filesystems that unifies **file-level and block-level** management
  - Contrast Linux: an ext4 filesystem is stored on a disk partition
    - (LVM2 allows more flexibility by "virtualising" hard disk partitions.)
- ZFS instead takes storage into a "pool" and allocates block extents and filesystems from that pool
  - By blurring block-level and file-level layers, ZFS can better optimise performance and resource usage
  - Installing a new hard disk can extend pool and all filesystems

Solaris ZFS integration with Zones

- ZFS was designed to support OS-level virtualisation
  - ZFS filesystems can be mounted hierarchically
  - (Commercial OSs often coordinate feature development…)

- A Zone's filesystem root is a sub-path of host filesystem
- On disk, Zones' data may be interleaved
  - … unlike isolated partitions on a conventional hard-disk
  - Advantage is sharing underlying redundancy, backup, deduplication and resource use across all zones' storage

Solaris Crossbow—virtualised networking

- We've seen VirtualBox / macOS net config. complexity
  - Labs involve NATing, NAT Networks, Host only networks, *etc*.
- Solaris Crossbow's virtualised networking support:
  - Provides all virtual machines / zones with IP presence
  - Allows host's resources to be flexibly shared
    - *e.g.*, bandwidth can be dynamically apportioned
- Solaris theme: flexible provisioning of host resources
  - *e.g.*, give host lots of disks; many NICs—can dynamically share

Back to Linux… since it powers the cloud

- Multifaceted Linux features often first componentised
  - Linux has a vast number of stakeholders
    - **Difficult to coordinate stakeholders** across different Linux parts
  - Effective OS-level virtualisation on Linux follows this practice
    - e.g., relying on separate cgroups and namespaces components
- We're setting the scene for Docker containers …
  - … but also explaining why there are so many different container systems, e.g., LXC, LXD, lmctfy, Docker, OpenVZ, Linux-vserver, Rkt, Singularity, …

Linux kernel namespaces (first release 2002)

- Namespaces only show processes subsets of resources
  - Two namespaces can **reuse the same IDs** (independently)
    - *e.g.*, user IDs, process IDs, filenames, *etc*.
  - Or a device only appears within one namespace
    - *e.g.*, network interfaces, *etc*.

- Namespaces used by container frameworks (~Docker)
  - Isolating containers' namespaces increases security
  - also simplifies software management (simpler resource alloc.)

Linux cgroups (first release 2007)

- A **control group** (cgroup) defines parameters about the resource use of a set of processes, *e.g.*:
  - limit total memory available to group of processes
  - indicate non-even share of device input/output priority
  - affect CPU scheduling to the group
  - cgroups also can assist accounting for resource use
- cgroups can facilitate starting / stopping processes
  - important for snapshot functionality