Virtualization in Linux

Thomas Petazzoni / Michael Opdenacker
Free Electrons
http://free-electrons.com/

Created with OpenOffice.org 2.x

Rights to copy

Document sources, updates and translations:
http://free-electrons.com/docs/virtualization

Corrections, suggestions, contributions and translations are welcome!

Why virtualization?

Virtualization can be used to implement

- Server consolidation: move several services into 1 physical server to reduce management and hardware costs. Possible to have different distribution versions on the same physical server.
- Disaster recovery: to quickly recover data and applications.
- Server security: implementing different services on isolated virtual machines.
- Replicating environments for software testing and development.
- Dedicated hosting: virtual private servers.

Virtualization approaches (1)

- Hardware emulation:
  Implements a full system on the host system. Can be with a completely different CPU. Unmodified guest systems.
- Native Virtualization:
  Full system too, but with the same CPU as on the host.
  Also supports unmodified guest systems.
- Hardware Enabled Virtualization:
  Takes advantage of CPU capabilities making it easier to implement virtualization and isolation (Intel and AMD processors)

Virtualization approaches (2)

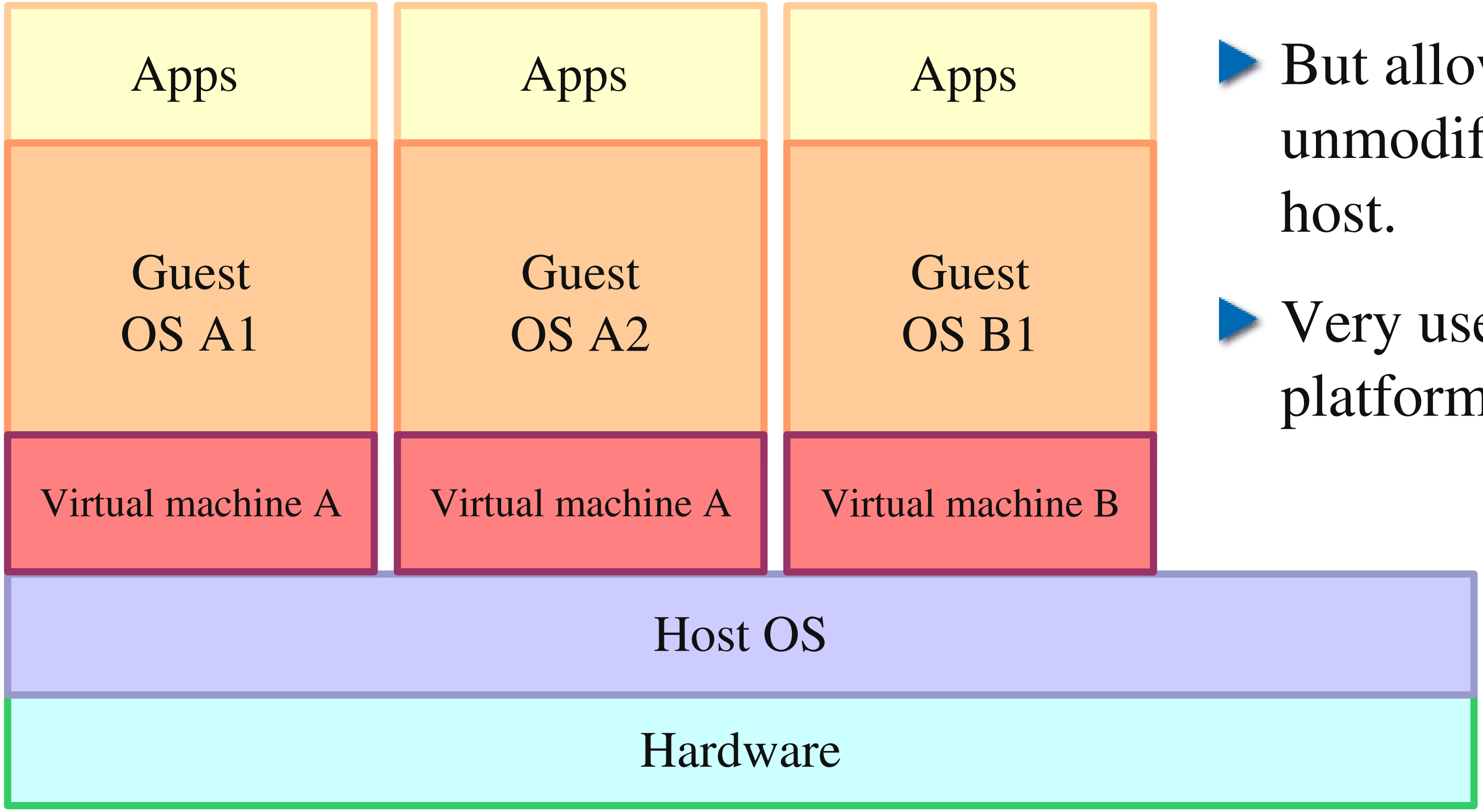
- Paravirtualization
  Not necessarily emulates hardware, but offers an API for the guest OS. The guest OS has to be modified to use this API instead of real hardware.
- Operating system level virtualization
  Same OS running guest and host systems, and offering isolation and virtualization.
- Processor virtualization:
  Running applications on a virtual processor (e.g. Java)

See http://en.wikipedia.org/wiki/Virtualization

Virtualization

Hardware emulation

Hardware emulation



Apps | Apps | Apps

Guest OS A1 | Guest OS A2 | Guest OS B1

Virtual machine A | Virtual machine A | Virtual machine B

Host OS

Hardware

- Slowest approach
- But allows to run any unmodified guest on the host.
- Very useful for cross platform development!

QEMU

http://qemu.org

- Very complete emulator, supporting complete systems (CPU + devices) in several systems: `x86`, `arm`, `powerpc`, `mips`, `m68k`...
- Uses dynamic code translation
- Pretty good performance:
  an emulated `arm` board can be faster than the real board!
- Emulating a PC on x86: no translation. Acts as a virtualizer, yielding very good performance.

Creating QEMU disk images

Did you know?

- You can create disk images for QEMU emulated systems, which can grow on demand.
- Example:

  ```
  > qemu-img create ubuntu804.img 4G
  ```

  Though the `ubuntu804.img` file is listed as 4G big, it only uses 0 bytes of disk space (at the beginning).
- If you manage multiple virtual machines with the same distribution, you can even just store the differences with a base image:

  ```
  >qemu-img create -b ubuntu804.img vm.img
  ```

Virtualization

Native virtualization

kqemu

- Kernel module allowing QEMU to reach close to native speed when emulating an x86 system on x86.
- Works by allowing to run userspace code directly on the host CPU.
- There is also an optional kernel emulation mode in which parts of kernel code can also be run on the host CPU.
- License: GPL

How to use kqemu

Instructions for Debian based systems (such as Ubuntu)

- Setup:

```
> sudo apt-get install module-assistant kqemu-common kqemu-source
> sudo module-assistant prepare
> sudo module-assistant auto-install kqemu-source
```

-
```
> sudo modprobe kqemu
> sudo chmod 666 /dev/kqemu
```

- Then run the `qemu` command.

Making kqemu always available

Instructions for Debian based systems

- To make sure that the kqemu module is always loaded: add `kqemu` to `/etc/modules`.
- Add a udev rule to make `/dev/kqemu` writeable by any user, by creating a `/etc/udev/rules.d/60-kqemu.rules` file containing:

```
KERNEL=="kqemu", NAME="%k", MODE="0666"
```

Tested on Ubuntu 8.04

VirtualBox

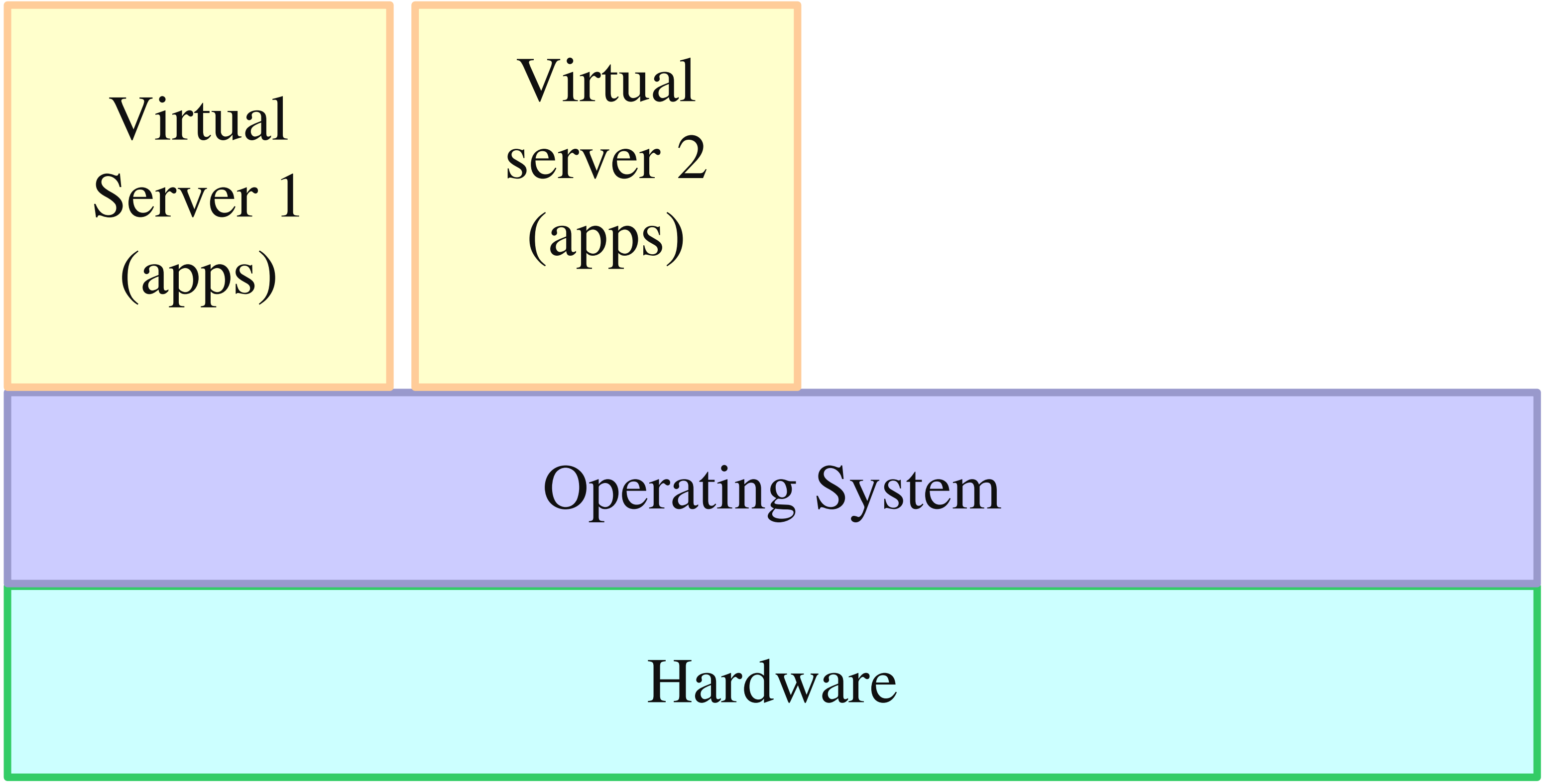
http://www.virtualbox.org/

- Another native virtualization solution.
- VirtualBox OSE: Open Source Edition (GPL license) available in mainstream GNU/Linux distributions
- VirtualBox: proprietary version. Adds a few features: Remote Display Protocol, USB, iSCSI.
- Based on QEMU's dynamic recompiler (LGPL). Achieves close to native speed too.
- The third most popular solution to run Windows in Linux.
- Acquired by Sun Microsystems in February 2008.

Operating system level virtualization

Operating system level virtualization

Virtual servers: isolated parts of the system in which applications are run



Linux-VServer

http://linux-vserver.org/

- Type: operating system level virtualization
- Implemented by *Security Contexts* providing chroot-like isolation. Allow to partition resources: file system, CPU time, network addresses and memory.
- Starting a virtual machine is just executing `/sbin/init` in each security context.
- Supports running unmodified GNU/Linux distributions.
- Available on most CPUs supported by Linux: `arm`, `mips`, `m68k`, `ppc`...

Linux-VServer advantages

- No emulation
  The virtual servers directly use the host system calls. No overhead processing virtual server network packets.
- Can use the same filesystem resources as the host.
- Virtual server processes scheduled by the host scheduler. Good for taking good advantage of multiple processors.
- Good performance. Very little overhead!
- Can be used in non x86 embedded systems!

Linux-VServer drawbacks

- Requires patches to the host kernel.
  Not available in mainstream kernels.
  Not available for the most recent kernels.
- Forced to have the same kernel as the host, sharing the same vulnerabilities and bugs.
- No virtualization: can't have specific routing or firewall setup for each virtual server.
- Can't implement virtual server specific I/O bandwidth allocation.
- Some hardware related system calls (e.g.: RTC) and /proc and /sys are not virtualized.

OpenVZ

http://wiki.openvz.org

- A containers based project
  Open Source core of Parallels Virtuozzo Containers, a commercial virtualization solution.
- Only supports Linux as the host and guest OS.
  Like Linux-Vserver, everything runs on the host kernel.
- Available as kernel patches and management tools.
- The project has brought a lot of improvements to containers support in the mainstream Linux kernel.

OpenVZ features (1)

Containers have their own:

- Files: filesystem, `/proc`, `/sys`
- Users and groups, in particular the root user
- Process tree: the init process has PID number 1
- Virtual network device (with IP address).
  Allows for specific firewall and routing rules.
- Devices:
  Possible to let a container access a real device.
- IPC objects: shared memory, locks, messages

OpenVZ features (2)

- 2-level disk quota
  Possible to have quotas inside a container
  with a global disk quota.
- 2-level scheduler
  First, a scheduler deciding which container should get the CPU
  Second, the ordinary Linux scheduler for processes inside the container.
- 2-level I/O scheduler
- And other per-container resources and limits (memory, etc.)
- Easy to share filesystem resources between containers (mass administration easier).

OpenVZ features (3)

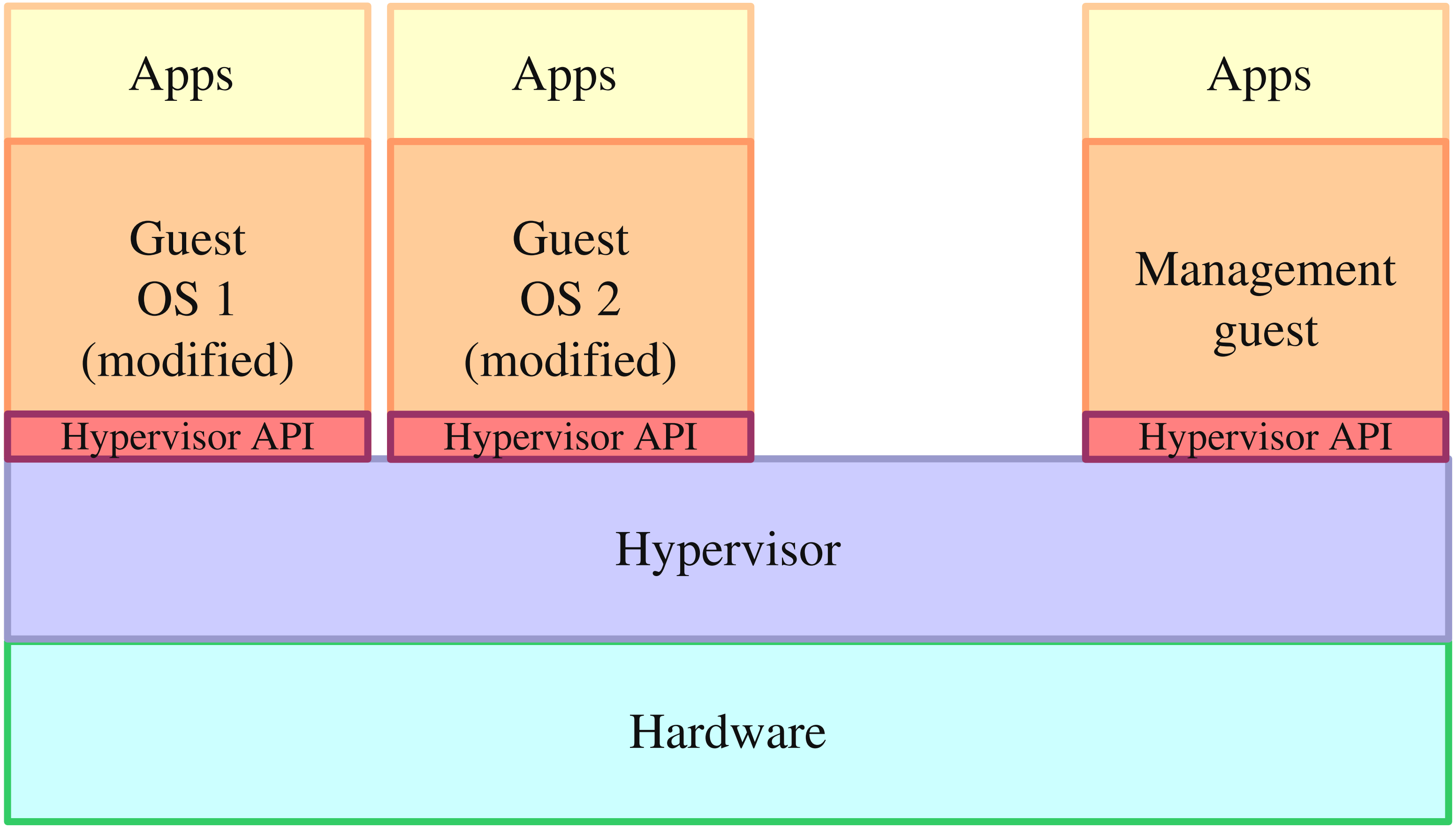
- Very low overhead due to virtualization
- Supports checkpointing: can freeze the state of a VZ, store it into a file, and restore it on another physical server.
- Easy to port to other architectures, as most of the code is platform independent. As of Linux 2.6.32:
  supports x86 (32 and 64 bit), arm, powerpc. and sparc.
- Main drawbacks:
  - Not mainstream yet, a great number of separate patches, but steadily converging with Vanilla Linux.

Virtualization

Paravirtualization

Paravirtualization

Virtual servers: isolated parts of the system in which applications are run



User Mode Linux

http://user-mode-linux.sourceforge.net/

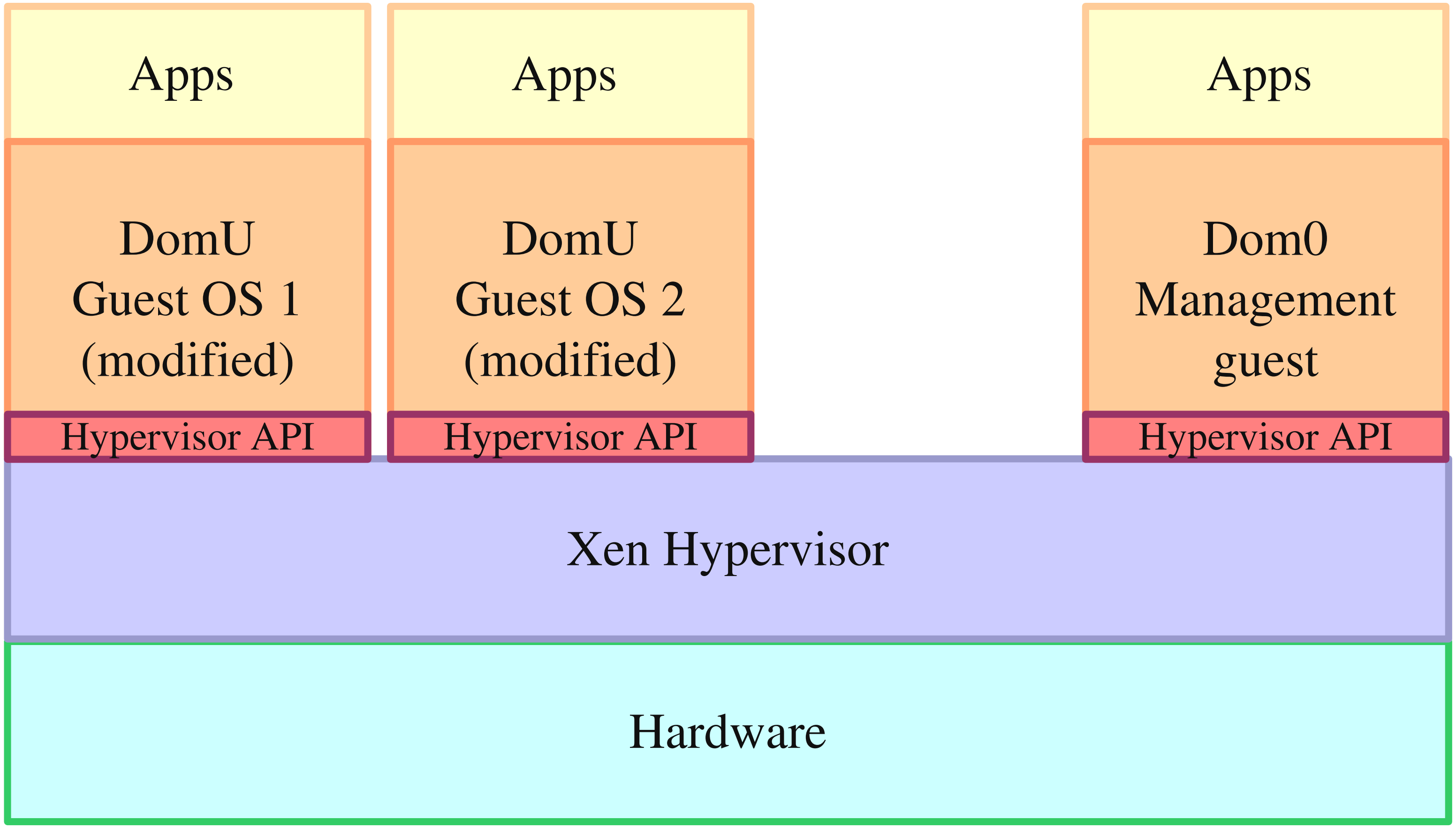
- Type: paravirtualization
- Port of the Linux kernel to its own system call interface.
  You can run a UML Linux kernel as a regular process!
- No special requirement for the host kernel.
  Can even run on old host Linux versions.
- Allowed the proliferation of virtual private Linux servers offerings on the Internet.
- Can use COW (copy on write) to share storage space between virtual machines.
- Possible to nest UML kernels!

User Mode Linux drawbacks

- Newer technologies perform better
- Supported architectures: x86 only
- Distribution filesystems have to be created manually (tweaks). Lack of ready-made filesystems.
- Still supported, but very little momentum.

Xen

http://www.xen.org/



Xen details

- Started as a research project at the University of Cambridge, led by Ian Pratt, who later found XenSource Inc.
- First public release in 2003. Quickly got very popular by showing very good performance compared to earlier solutions.
- As a paravirtualization solution, requires guest OSes to be modified.
- Part of Xen was merged in Linux 2.6.23, allowing a kernel to boot in a paravirtualized environment under the Xen hypervizor.
- October 2007: XenSource acquired by Citrix Solutions.
- Supported architectures: `x86` (32 and 64 bit), `ia64` and `ppc`. Some people have started a port to `arm` (not ready yet!)

Dom0

The first guest OS automatically started by the hypervizor

- Has special priviledges to administrate the hypervizor, create, control and stop other virtual machines.
- Has direct access to the physical hardware.
- Usually uses the same kernel as other Linux guests.
- Should be very well protected, and shouldn't run any service that could expose it to attacks, as this would compromise all guest OSes.

DomU

Any other virtual machine created from Dom0

- Originally can only run modified guest OSes.
  Of course, applications don't need to be modified!
- Since Xen 3.0, possible to run unmodified versions of Windows (and Linux of course), if the hardware has hardware enabled virtualization.
- Can be suspended, rebooted or even migrated to another physical server.

Xen resources

- HowtoForge:
  A very good tutorial covering Xen installation and basic usage.
  http://howtoforge.com/ubuntu-7.10-server-install-xen-from-ubuntu-repositories
- Xen community home page
  http://www.xen.org/
- Xen documentation:
  http://www.xen.org/xen/documentation.html

Virtualization

Hardware enabled virtualization

Hardware enabled virtualization

- Requires virtualization support in hardware: Intel-VT or AMD-V (Pacifica) extensions.
- Allows to run unmodified guest OSes
- Free Software solutions:
  - Xen since 3.0
  - KVM

KVM - Kernel based Virtual Machine

http://kvm.qumranet.com/

- Only for Intel-VT and AMD-V processors.
- Available as a kernel module to load (and as a system call in the future?). Included in standard Linux since 2.6.20.
- Each virtual machine has virtualized hardware, emulated by qemu: a network card, disk, graphics adapter, etc. Graphics are supported!
- Each virtual machine is started as a standard user-space process, running a modified version of qemu. Such processes can then monitored, prioritized and managed like any other processes.
- “The ultimate hypervisor is the Linux kernel”. No need to implement yet another hypervisor with a scheduler, memory management, etc.

KVM features (1)

- Currently supports `x86`, `ppc`, `s390` and `ia64`.
- Supports live migration from one physical server to another (this is indeed supported by qemu).
- Networking implementing in exactly the same way as in qemu. Easy!
- Similarly, supports the same disk images as qemu. In particular, they can grow automatically according to space actually in use.

KVM features (2)

- Virtual machines can be run by unpriviledged users.
- Virtual machines can be started with a given maximum amount of RAM (`-m` qemu option), but don't necessarily use all of it.

KVM examples

- Creating a disk image:

```
qemu-img create disk.img 5G
```

- Loading KVM modules:

```
modprobe kvm
```

`modprobe kvm_intel` (or `modprobe kvm_amd`)

- Creating and installing the virtual machine:

```
kvm -hda disk.img -cdrom os.iso -m 512 -boot d
```

- Restarting this machine after installation:

```
kvm -hda disk.img -m 512 -boot d
```

Useful KVM links

- KVM FAQ
  http://kvm.qumranet.com/kvmwiki/FAQ
- KVM on Wikipedia
  http://en.wikipedia.org/wiki/Kernel-based_Virtual_Machine
- Qemu documentation:
  http://bellard.org/qemu/qemu-doc.html

Summary - Compared capabilities

	Virtualization type	Performance compared to host	Multiple guest OS support?	Unmodified guests	Non x86 host and guest support
QEMU	Hardware emulation	10-20%	Yes	Yes	Yes
QEMU with kqemu	Native virtualization	Near native	Yes	Yes	No
KVM	Hardware enabled virtualization	Near native	Yes	Yes	No
Xen with Intel-VT or AMD-V	Hardware enabled virtualization	Near native	Yes	Yes	No
Xen	Paravirtualization	Near native	Yes	No	No (`ppc` and `ia64` ports in progress)
User Mode Linux	Paravirtualization	Near native	No	No	No
Linux-VServers	Operating system level virtualization	Native	No	Yes	Yes
OpenVZ	Operating system level virtualization	Native	No	Yes	No (but possible)

Details: http://en.wikipedia.org/wiki/Comparison_of_virtual_machines

References

- Virtual Linux, by IBM:
  http://www.ibm.com/developerworks/library/l-linuxvirt/
- Comparison of virtual machines (free and proprietary):
  http://en.wikipedia.org/wiki/Comparison_of_virtual_machines

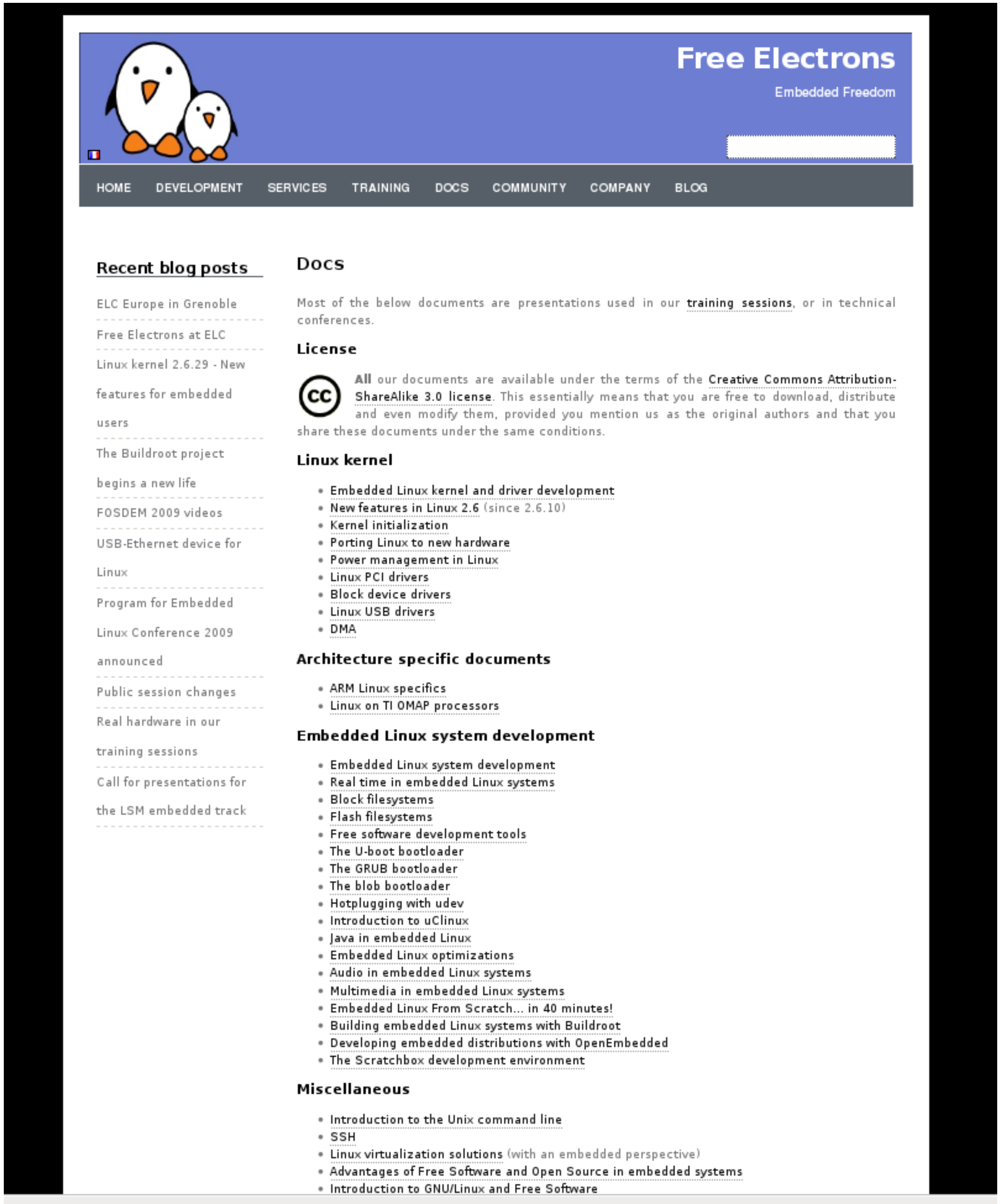
Ubuntu Jeos

http://ubuntu.com/products/whatisubuntu/serveredition/jeos

- Pronounce it “Juice”
  A “Just Enough OS” for virtual appliances.
- Smaller footprint than a standard server distribution
  - Less packages (less stuff to remove from a server distro).
    Useless packages though: wireless-tools, wpasupplicant, pcmciautils, usbutils, hardware drivers (sound).
  - Fewer updates, less maintenance work.
- Optimized for VMWare and KVM
- Ubuntu Jeos 8.04: free of cost and long term support (5 years)

See also https://help.ubuntu.com/community/JeOS

Related documents



Recent blog posts

- ELC Europe in Grenoble
- Free Electrons at ELC
- Linux kernel 2.6.29 - New features for embedded users
- The Buildroot project begins a new life
- FOSDEM 2009 videos
- USB-Ethernet device for Linux
- Program for Embedded Linux Conference 2009 announced
- Public session changes
- Real hardware in our training sessions
- Call for presentations for the LSM embedded track

Docs

Most of the below documents are presentations used in our training sessions, or in technical conferences.

License

All our documents are available under the terms of the Creative Commons Attribution-ShareAlike 3.0 license. This essentially means that you are free to download, distribute and even modify them, provided you mention us as the original authors and that you share these documents under the same conditions.

Linux kernel

- Embedded Linux kernel and driver development
- New features in Linux 2.6 (since 2.6.10)
- Kernel initialization
- Porting Linux to new hardware
- Power management in Linux
- Linux PCI drivers
- Block device drivers
- Linux USB drivers
- DMA

Architecture specific documents

- ARM Linux specifics
- Linux on TI OMAP processors

Embedded Linux system development

- Embedded Linux system development
- Real time in embedded Linux systems
- Block filesystems
- Flash filesystems
- Free software development tools
- The U-boot bootloader
- The GRUB bootloader
- The blob bootloader
- Hotplugging with udev
- Introduction to uClinux
- Java in embedded Linux
- Embedded Linux optimizations
- Audio in embedded Linux systems
- Multimedia in embedded Linux systems
- Embedded Linux From Scratch... in 40 minutes!
- Building embedded Linux systems with Buildroot
- Developing embedded distributions with OpenEmbedded
- The Scratchbox development environment

Miscellaneous

- Introduction to the Unix command line
- SSH
- Linux virtualization solutions (with an embedded perspective)
- Advantages of Free Software and Open Source in embedded systems
- Introduction to GNU/Linux and Free Software

All our technical presentations
on http://free-electrons.com/docs

- Linux kernel
- Device drivers
- Architecture specifics
- Embedded Linux system development

How to help

You can help us to improve and maintain this document...

- By sending corrections, suggestions, contributions and translations
- By asking your organization to order development, consulting and training services performed by the authors of these documents (see http://free-electrons.com/).
- By sharing this document with your friends, colleagues and with the local Free Software community.
- By adding links on your website to our on-line materials, to increase their visibility in search engine results.

Free Electrons

Our services

Linux kernel

Linux device drivers
Board support code
Mainstreaming kernel code
Kernel debugging

Embedded Linux Training

All materials released with a free license!

Unix and GNU/Linux basics
Linux kernel and drivers development
Real-time Linux, uClinux
Development and profiling tools
Lightweight tools for embedded systems
Root filesystem creation
Audio and multimedia
System optimization

Custom Development

System integration
Embedded Linux demos and prototypes
System optimization
Application and interface development

Consulting and technical support

Help in decision making
System architecture
System design and performance review
Development tool and application support
Investigating issues and fixing tool bugs