CI583: Data Structures and Operating Systems
Virtualisation
Distributed systems

Virtualisation

A virtual machine (VM) is an operating system hosted within and running on top of another (non-virtual) OS.

Virtualisation

In the last ten or 15 years, virtualisation has become an increasingly important topic, important enough for Intel and AMD to extend the x86 architecture to give hardware support for it.

Image ©http://lifehacker.com/

Virtualisation

In the last ten or 15 years, virtualisation has become an increasingly important topic, important enough for Intel and AMD to extend the x86 architecture to give hardware support for it.

Image ©http://lifehacker.com/

VMs are now an essential component of datacentres and other network infrastructures, and are relied upon by developers and "ordinary" users alike.

Virtualisation

Virtualisation is actually quite an old technology.

Like many key concepts in the world of operating systems, it was developed in the 1960s.

They were intended as a solution to the problem of multiuser timesharing systems – one way of providing users with a share of systems resources is to give them each their own copy of the operating system (IBM's `CMS`).

This is taking the notion of isolating each user's processes to the extreme, and can be seen as a form of the microkernel approach.

Virtualisation

The kernel, in this case, is a virtual machine monitor (VMM), whose main responsibility is scheduling, switching contexts between the virtual machines and making sure each gets a fair amount of processor time.

In more modern VM systems the VMM is known as the host or hypervisor, and the VMs are known as guests.

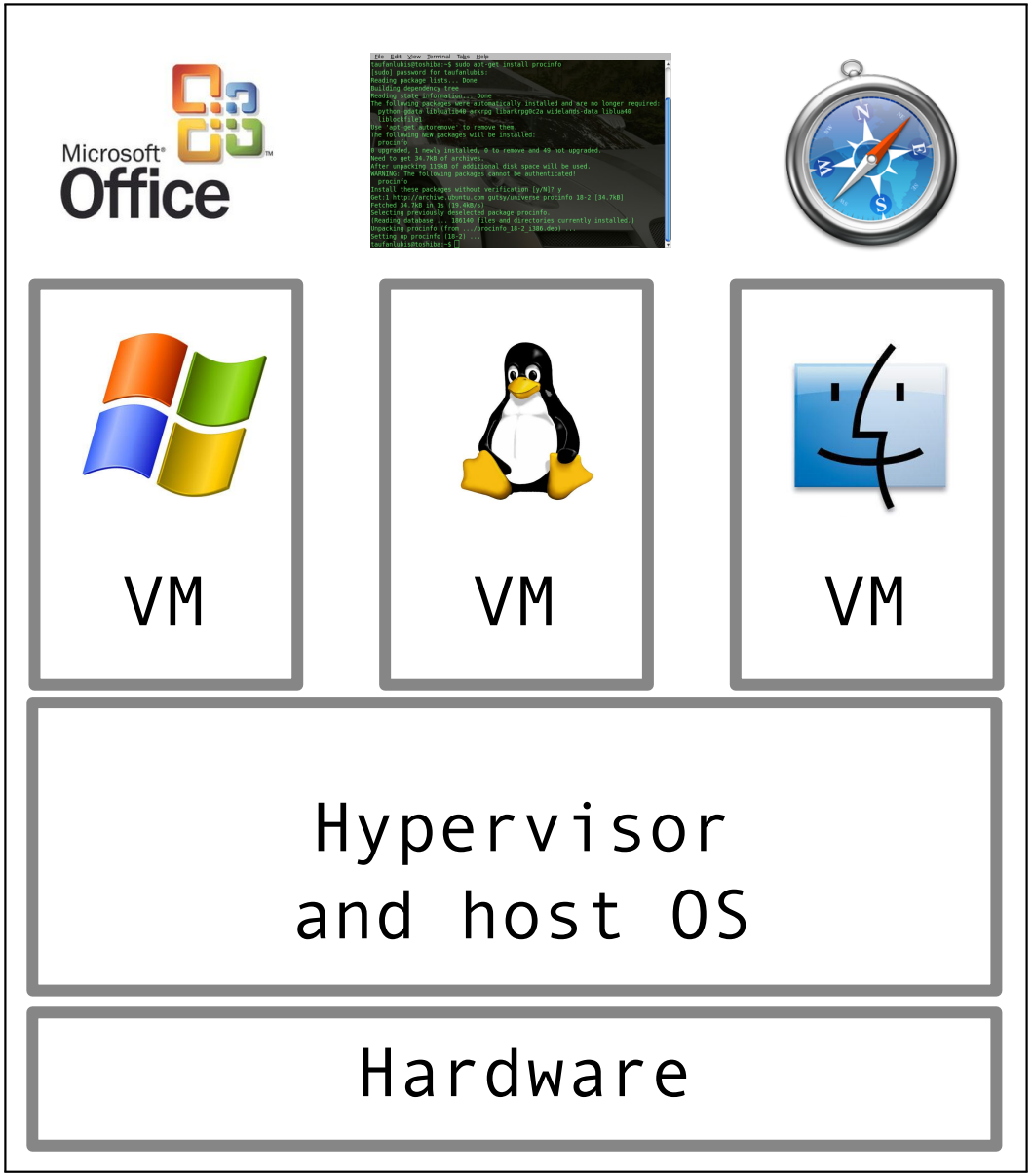
Virtualisation

The host manages fair access to resources (IO devices, processor time, etc), whilst each guest has its own image of an entire OS, including a file system, IO modules, scheduler, hardware address translation facility etc – each of these is virtual.

(In the literature around VMs, terms like *hypervisor* are used in several ways, sometimes to mean the host OS and sometimes the VMM.

These might not be the same thing. We use hypervisor to mean the VMM.)

Virtualisation



Virtualisation

VM architecture has become important because it is so useful:

1. Run multiple platforms on the same box: main selling point for developers and home users. No longer need dedicated hardware to run an app that is only available for a certain OS, or to develop and test programs for that OS. Especially true if you are developing an OS.

   Allows the creation of emulators (e.g. for Android) that present a true image of a system.

Virtualisation

3. Run multiple, isolated servers on the same box: main selling point in network infrastructure. This allows for server consolidation and isolation (e.g. restart the web server whilst keeping the db server running, even though they are VMs hosted on the same machine).

   This is economical and makes it easier to back up and relocate servers. E.g. MS Sharepoint guidelines indicate separate servers for AD, web portal, SQL Server, file server etc. These need not be physical servers. Also very useful in shared hosting – give each user their own installation with out needed to provide physical machines.

Approaches to virtualisation

There are two ways we might implement VMs: pure virtualisation or para-virtualisation.

Using pure virtualisation, each VM runs entirely in userland. It is ordinary software that interacts with the host OS like any other process.

However, the processor that the VM sees is the real one. The guest can execute instructions directly, generate and handle its own interrupts, page faults etc. This is the model used by `VMWare`.

Approaches to virtualisation

The potential problem with this approach is that when the VM is in system mode (handling a page fault, for instance), it has complete control over the host. In practice, this is insecure and complex.

`VMWare` solved the problem by translating (at the machine code level) privileged instructions emanating from a VM into something that was definitely safe to run.

Approaches to virtualisation

Since then, manufacturers of processors have recognised the importance of virtualisation and enhanced the x86 architecture to support new modes.

As well as privileged and user, we now have virtual privileged mode and virtual user mode.

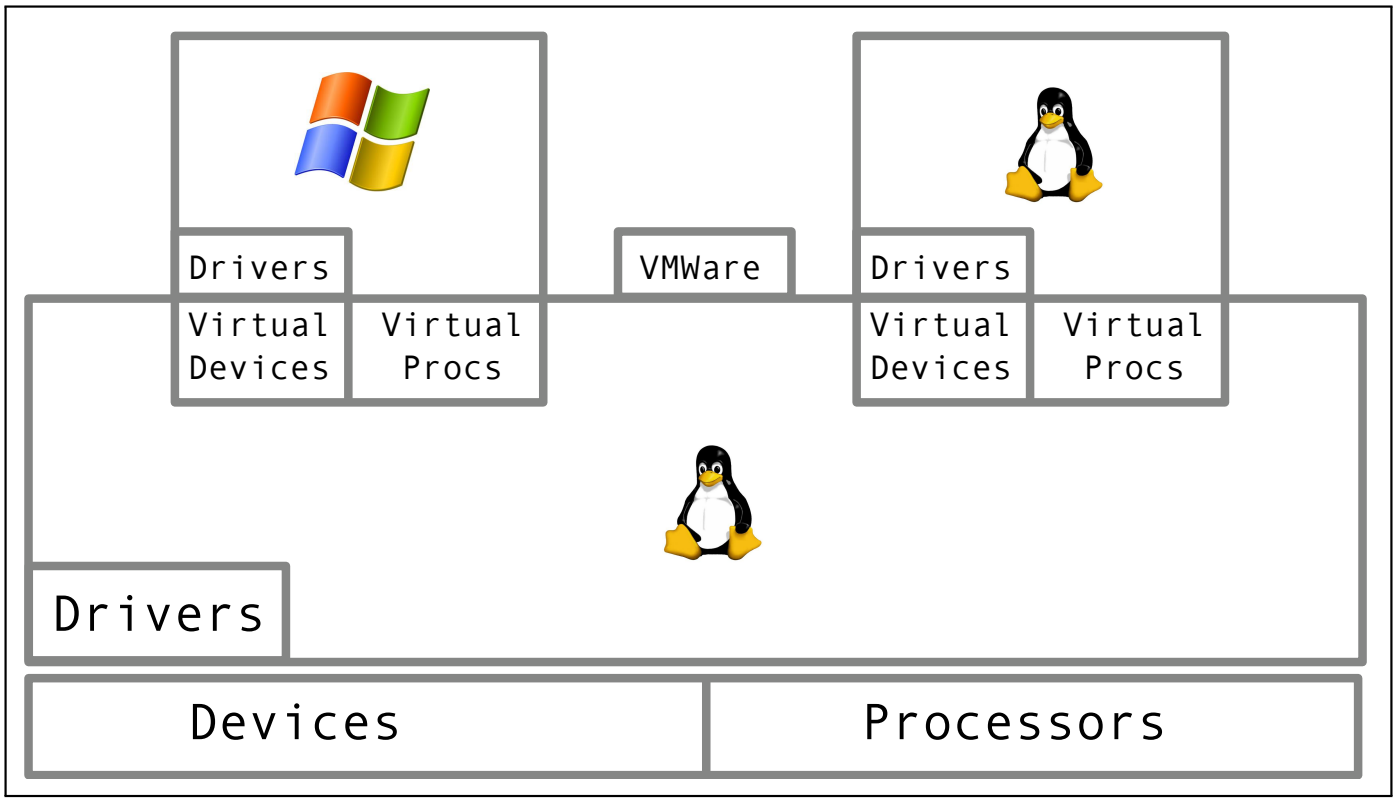
Approaches to virtualisation

When a VM appears to itself to be in privileged mode, it is really in virtual privileged mode, running in userland on the host OS, with the VMM translating its system calls into real ones.

The capabilities of virtual privileged mode are no longer complete – e.g. the page fault handling routine of a VM can only modify a page table if it was previously assigned to that VM.

The VMM needs to manage a bridge between the VM's virtual IO devices, page table, etc, and the real ones.

Approaches to virtualisation



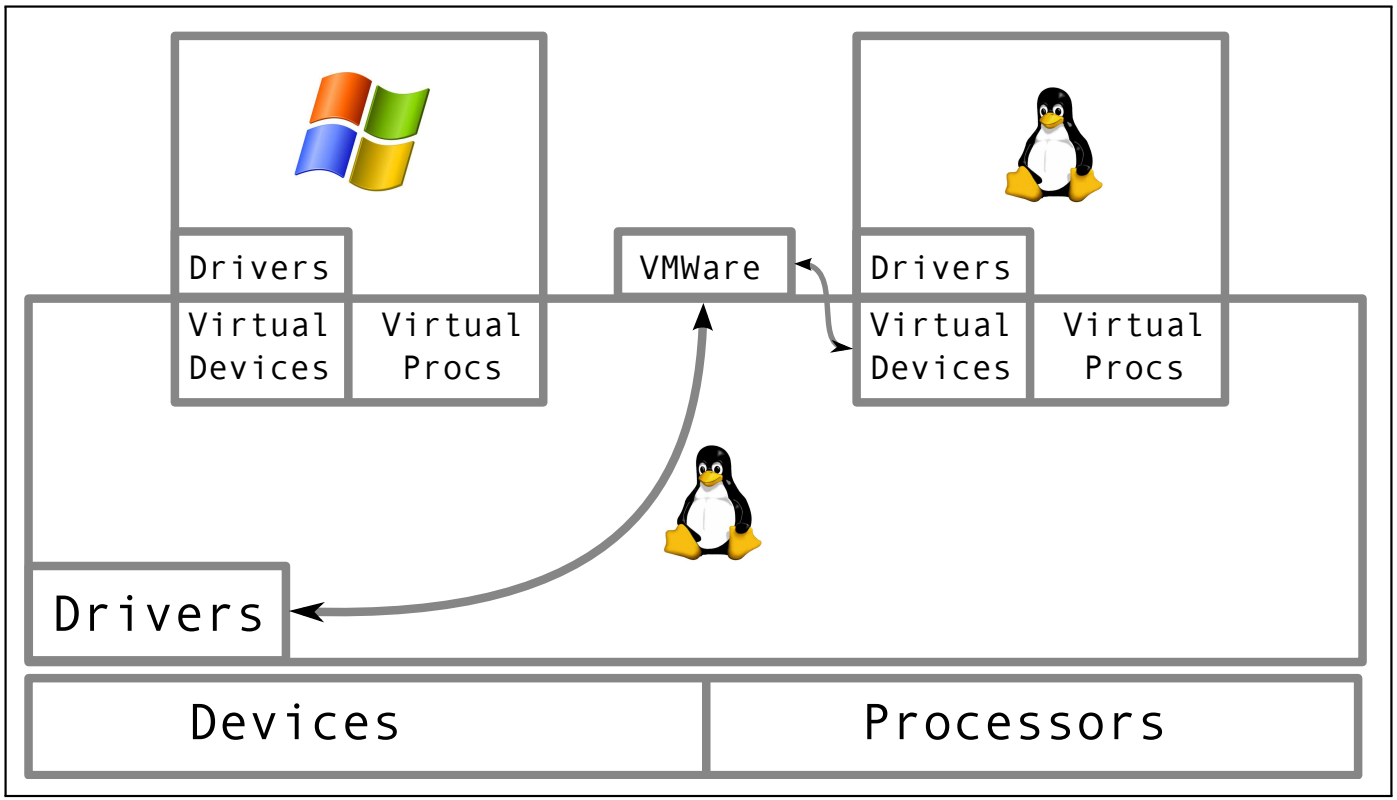
Approaches to virtualisation

The activity of each VM is time-sliced, so that it can be handled by the scheduler. Thus, its own hardware clock is a virtual one which only runs when the VM is running.

However, the VM also needs a way to discover the value of the real hardware clock to handle actions that depend on it, like timeouts, or when to retransmit a network packet.

This is done in practice by using system calls that would never otherwise be seen on the host, so-called hypervisor calls.

Approaches to virtualisation



Approaches to virtualisation

Thus, VMs appear as processes. When the host OS switches context from one VM to another, this is an extremely expensive operation – the context of a VM process includes the entire context of an OS.

Similarly to virtual privileged mode, x86 extensions now allow VMMs to deal with real and virtual virtual memory locations(!).

Approaches to virtualisation

The x86 architecture now includes a special mode in the hardware address translation facility (which maps VMLs to page frames) that maps the VMLs used by a VM (which are virtual, of course) to real VMLs.

This gets rid of the need for a VMM to manage its own mapping between the two. This is called extended page translation.

See Intel article on this: `http://tx0.org/4v1`.

Approaches to virtualisation

Problems with pure virtualisation:

1. Performance issues, such as context switching and emulating privileged instructions.
2. Duplication of effort: when running n VMs, there are $n + 1$ schedulers running, $n + 1$ virtual memory managers, etc.
3. Difficulty/messiness of access to non-virtualisable functions, like hardware clock.

Para-virtualisation

Para-virtualisation solves many of these problems by modifying the OS or running the OS entirely within a VMM, rather than as a normal process.

An example of the former approach is `Denali` (University of Washington), which runs specially modified lightweight Linux-based VMs.

In particular, the VMs have a simplified IO architecture, which makes context switching less expensive. `Denali` is designed to run 100s or 1000s of VMs on a single host.

Approaches to virtualisation

More widely known systems such as `VirtualBox` and `Xen` use a mixture of pure and para-virtualisation by running an unmodified OS but not as a single (real) OS process, but entirely within a VMM.

The VMM handles all scheduling issues, access to virtual privileged mode (needed for page fault handling etc), and so on.

Each VM is entirely isolated and runs in userland, but still presents an image of a complete OS. It has no more control over the host OS than any other process.