

Introduction to Advanced Operating Systems and μ Kernels

Luca Abeni

luca.abeni@santannapisa.it

February 25, 2021

Advanced Operating Systems

- Assumption: you know the basics of Operating Systems
 - Here, we study more structures and architectures...
 - ...To be used for performance, predictability, security, or for some special reason
- Examples of special-purpose OSs:
 - Real-Time Operating Systems
 - Virtual Machines (hypervisors, unikernels, ...)
- Example: cloud computing, embedded systems, control systems, ...

Specializing the OS...

- Different application fields have special requirements
 - Efficient virtualization of CPU and or devices
 - Security
 - Small size / small memory footprint
 - ...
- A “textbook OS” is clearly not enough
- General purpose OSs can do many things... But not everything in the best way!
 - Example: what does “performance” mean?
Throughput, or latency???

Real-Time Operating Systems

- Real-Time operating system (RTOS): OS providing support to Real-Time applications
- Real-Time application: the correctness depends not only on the output values, but also on the time when such values are produced
- Operating System:
 - Set of computer programs
 - Interface between applications and hardware
 - Control the execution of application programs
 - Manage the hardware and software resources

Different Visions of an OS

- An OS manages resources to provide services...
- ...hence, it can be seen as:
 - A Service Provider for user programs
 - Exports a programming interface...
 - A Resource Manager
 - Implements schedulers...

Operating System as a Resource Manager

- **Process** and **Memory** Management
- **File** Management
 - VFS
 - File System
- Networking, Device Drivers, Graphical Interface

Resources must be managed so that
real-time applications are served properly

Operating System Services

- Services (Kernel Space):
 - Process Synchronisation, Inter-Process Communication (IPC)
 - Process / Thread Scheduling
 - I / O
 - Virtual Memory

Specialised API?

Resource Management Algorithms

- Resource Manager (device drivers, ...)
 - Interrupt Handling
 - Device Management
 - ...

OS Structure?

Real-Time Systems: What???

- Real-Time application: the time when a result is produced matters
 - a correct result produced too late is equivalent to a wrong result, or to no result
 - characterised by **temporal constraints** that have to be respected
- Example: mobile vehicle with a software module that
 1. Detects obstacles
 2. Computes a new trajectory to avoid them
 3. Computes the commands for engine, brakes, ...
 4. Sends the commands

Real-Time Systems: What???

- If the commands are correctly computed, but are not sent in time...
- ...The vehicle crashes into the obstacle before receiving the commands!
- Examples of temporal constraints:
 - must react to external events in a predictable time
 - must repeat a given activity at a precise rate
 - must end an activity before a specified time
- Temporal constraints are modelled using the concept of *deadline*

Real-Time & Determinism

- A Real-Time system is not just a “fast system” . . .
- speed is always relative to a specific environment!
- Running faster is good, but does not guarantee a correct behaviour
 - It must be possible to *prove* that temporal constraints are **always respected**
 - Running “fast enough”
 - . . . \Rightarrow worst-case analysis

Throughput vs Real-Time

- Real-Time systems and general-purpose systems have different goals
 - General-purpose systems are optimised for the “most common” or “average” case → fast systems
 - Real-Time systems only care about the worst case
- In general, fast systems tend to minimise the average response time of a task set ...
- ... While a real-time system *must* guarantee the timing behaviour of RT tasks!

Virtual Machines

- Virtual Machine: efficient, isolated duplicate of a physical machine
 - Execution environment essentially identical to the physical machine
 - Programs only see a small decrease in speed
 - A “monitor” or “hypervisor” is in full control of physical resources
- Programs running in a VM **should** not see differences respect to real hw
- Virtualization should be efficient
- Programs should not be able to access resources outside of the VM

VMs and OSs

- How is an OS related to Virtual Machines?
 - The OS should provide support for the Virtual Machine Monitor / hypervisor
 - The OS *could* be optimized to run inside a VM
- OS support for virtualization (as host or as guest)
 - Impact on resource management
 - Impact on the exposed features
 - Impact on the I/O devices support
- Impact on the OS architecture?
 - Host: type-1 hypervisors, μ -kernel systems
 - Guest: library OSs, unikernels, vertically structured OSs

CPU Virtualization

- First idea: simulate the CPU hw in software
 - Software implementation of an abstract machine implementing the fetch-decode-execute-(write) cycle
 - **Fails** the **efficiency** requirement!!!
- Other idea: directly execute the virtualized instructions on the CPU
 - Virtual ISA: exact copy of the host ISA
 - Might fail the third (VMM is in control) requirement
 - Limited to unprivileged instructions (with VMM executing at a high privilege level)
 - What to do for privileged instructions?

Virtualizable CPU Architectures

- The monitor should be able to “intercept” some machine instructions
 - Some kind of trap / exception / software interrupt must be generated
 - Not always possible (think about x86 ring 0)
- The CPU must provide some support for full virtualization
 - “More than supervisor” mode → hypervisor mode
 - Introduce two operating modes: “root mode” and “non-root mode”; non-root mode can only modify a shadow copy of the CPU privileged state
 - ...

OSs for Virtualizable Architectures

- Virtualizable ISA: how to use it?
 - VMM or hypervisor responsible for managing VMs and other resources
 - Re-invent an OS, or using an existing one?
- OS support for hypervisors
 - Hosted hypervisor
 - Dom0
 - ...
- Difference between a hypervisor and a μ -kernel???
 - Are we reinventing an old idea?
 - ...And, what are μ -kernels, after all???

ParaVirtualization

- So, CPU virtualization can be easy and efficient
 - Provided that the ISA is virtualizable
 - Provided host OS support / hypervisor
- What about I/O devices?
 - Virtualizing real hardware can be complex and inefficient
 - Idea: device passthrough
 - Other possibility: paravirtualization
- Paravirtualization: the guest knows that it is running in a VM
 - Memory buffers can be (securely) shared between guest and host
 - ...

Traditional OS Structure

- OS: Set of computer programs interfacing user applications with the hardware
 - Kernel: part of the OS running at high privilege level
 - Traditionally includes a lot of things (drivers, network stack, fs, ...)
 - Do we really need high privileges for all of this?
- Special-purpose OSs often propose different structures
 - For virtualization, hypervisor running “below the kernel”
 - For real-time, dual-kernel approach
 - For security, reduced trusted code base...

μ Kernels - The Idea

- Basic idea: simplify the kernel
 - Reduce to the number of abstractions implemented by the kernel
 - Address Spaces
 - Threads
 - IPC mechanisms (channels, ports, etc...)
 - Most of the “traditional” kernel functionalities implemented in user space
 - Even device drivers can be in user space!

μ Kernels and Servers

- Interactions via IPC (IRQs to drivers as messages, ...)
- Servers: US processes implementing OS functionalities
 - OS kernel as a single user-space process:
Single-server OSs
 - Multiple user-space processes (a server per driver, FS server, network server, ...):
Multi-server OSs