Unikernels

COSC349—Cloud Computing Architecture

David Eyers

Learning objectives

- Define the term '**unikernel**'
- Contrast degree of specialisation within VM types, e.g., full hardware VMs, Docker containers, and unikernels
- Enumerate good & bad points about unikernels
- Sketch some existing **unikernel projects**
- Describe the typical **role of VMM** in unikernel systems

Specialisation versus generalisation

- We've seen **styles of virtualisation** ranging from:
  - general purpose: VirtualBox—full hardware virtualisation
  - less general purpose: Vagrant—for developers
  - specific purpose: Docker—VMs do one specific job (usually)
- Docker containers' Unix shells used in emergencies
  - Shouldn't need general-purpose OS interactions
- Unikernels are an even more specific form of VM
  - e.g., **no Unix shell** at all, possibly **no multitasking**, …

What can be stripped from a Docker image?

- Some examples of the types of stripping down possible:
  - Assume **never need to install software**: no package system
  - Assume that we don't need to use a shell: **no shell**
    - This means the OS has to start the application directly
  - Assume configuration can be "baked in": **no filesystem**
  - Assume **no operating system driver** changes
- VM ends up behaving like an executable program
  - … except it contains what it needs of its own operating system

Unikernels

- Unikernels are OS kernels that **can only do one job**
  - This is not a new idea: Library OSs involve the same notion
- Benefits:
  - Extremely fast **boot times**
  - Very small **memory overhead**
  - Small surface area in terms of potential **security** problems
- Downsides:
  - **Building / changing unikernels** often expensive (time+resource)

Present-day unikernel viability

- Unikernels don't run on bare metal, instead **on VMMs**
- Unikernels' "hardware" is typically paravirtual devices
  - Works fine for network, block storage and simple console I/O
  - Real hardware device drivers are within VMM (or Xen dom0)
  - (Not including device drivers is practical rather than technical)
- Many applications can be built **using HTTP(S), alone**
  - e.g., VMs offering and consuming micro-services
  - VM does not have persistent state
  - Interact with external servers to effect **persistent storage**

Challenge of rebuilding unikernels

- Run-time aspects become **build-time dependencies**
  - Changing anything can involve significant compile+link effort
  - Link process made cheaper in OSs by dynamic link libraries
  - OS libraries can be updated independently of program code
- Compilers usually rebuild quickly from intermediate files
  - Note the typical conflicting priorities of compiler design:
    - Speed of executable, size of executable, speed of compilation, …
- Notion of "**cloud native**" software is spreading
  - Over time, expect changes in code building environment

Lots of unikernel projects in recent years

- ClickOS, Clive, Drawbridge, HaLVM, HermitCore, OSv, IncludeOS, LING, MirageOS, Rumprun, runtime.js, UniK
  - Many of these projects are **programming-language led**
- Appealing route for doing clean-slate OS design
  - However so much OS-code is C/C++; can't afford to start over
  - So working above the VMM is a good compromise
- Many are **functional PLs**: Haskell, Erlang, OCaml, …
  - There typically won't be userspace / kernel division in unikernel
  - Thus want "safe" programming languages

LING—an Erlang microkernel framework

- Erlang language popularised actors & supervisor trees
  - Ericsson telephone exchange software—want zero downtime
  - Live software updates
  - Good language for microservices
- **Erlang-on-Xen**—https://github.com/cloudozer/ling
  - Mitigates vulnerabilities: read-only filesystem, no OpenSSL
  - Responsive: 100ms boot to shell
  - Doesn't leave processes waiting for incoming network requests
    - Can boot unikernels fast enough to start them on demand

IncludeOS

- IncludeOS is implemented in C++, and supports C/C++
- Event-driven approach to interacting with OS
  - Similar to the approach of Node.js—asynchronous callbacks
  - **Cooperative multitasking** is a common unikernel design:
    - Avoids need for task schedulers, not least if VMM already has one
- Design priorities:
  - **Security**: unikernel image is immutable; used components only
  - **Size**: typical applications use 2–3MB; only need 4–5MB RAM
  - **Performance**: no context switches; whole system optimisation

MirageOS

- **Uses OCaml**: functional, OO, statically typed language
  - Impure functional language—allows side-effects and state
    - return value of max(Set) should just depend on inputs
    - a function like malloc() won't return the same value for same parameters
  - OCaml has been shown to outpace C code in some contexts
    - e.g., when OCaml can optimise code to avoid copying of memory
- MirageOS boots on Xen—OCaml Labs & Xen teams overlap
  - Early versions had **no filesystem**
  - … but it's practical for REST over HTTPS to effect network apps
  - Example application: **self-hosting website**