Unikernelized Linux

Tiejun Chen <tiejunc@vmware.com>
VMware China R&D Advanced Technology Center

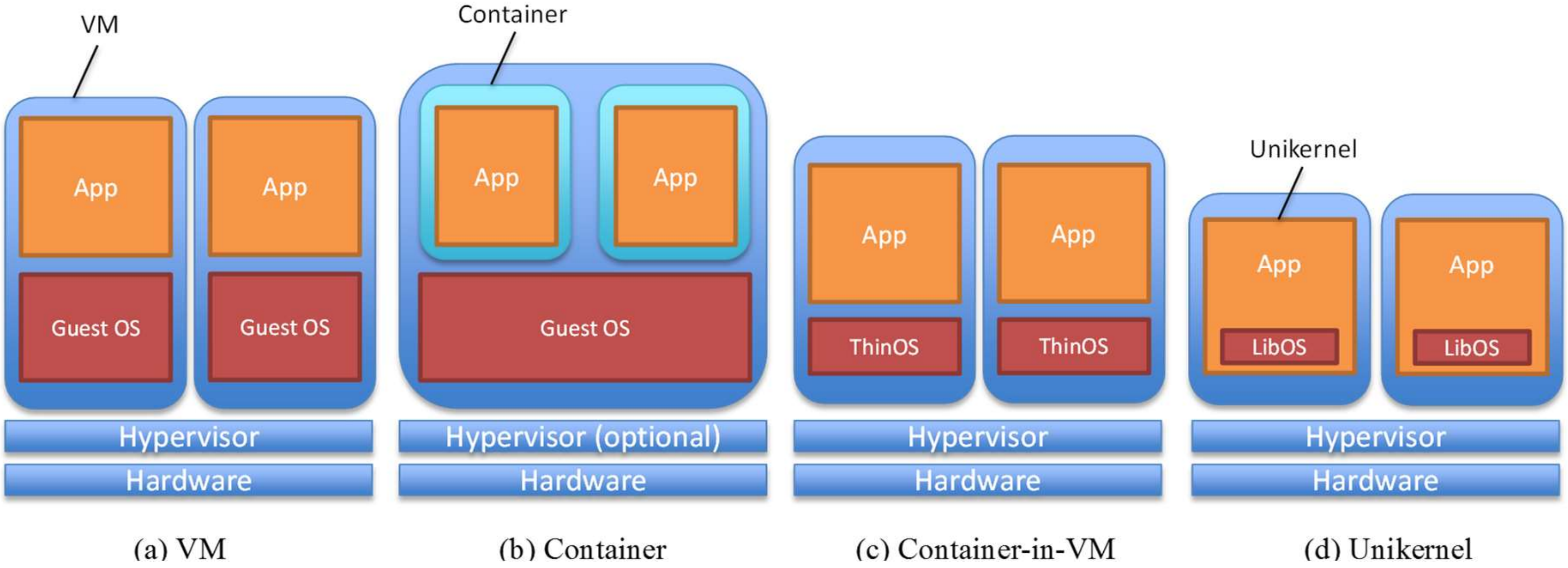
Warning ☺

This is our own exploration of unikernels.

This is not a roadmap or commitment from VMware.

Background

- **Linux container technologies like Docker dominate dramatically**
  - An efficient but easy way to carry out applications to provide cloud services in the different cases.
- **A new technology called unikernels is beginning to attract our attention**
  - Unikernels are developing a variety of new approaches to deploy cloud services.



Major Existing Unikernels

- **Unikernels projects**
  - MirageOS, ClickOS, Clive, HaLVM, LING, Rump Kernels, OSv, Unik, Solo5 Unikernel, Drawbridge
- **Unikernels solutions**
  - Docker
    - Hyperkit/VPNkit
    - Moby/Linuxkit
  - Mikelangelo
    - Improving Responsiveness and Agility of HPC Cloud Infrastructure
  - NFV
    - Unikernels based NFV architecture

Unikernel Definition and Types

- **Definition**
  - Unikernels are specialised, single address space machine images constructed by using library operating systems. *--Wiki*
- **Types**
  - General purpose unikernels
    - A library that derives from a generally designed OS kernel
    - Works for apps that follow some mature speculations (e.g. POSIX, or glibc)
    - Example: Rumprun, OSv, ClickOS and Drawbridge
  - Language specific unikernels
    - A library of a programming language that includes all OS functionalities
    - Works for apps written in specific languages only
    - Example: MirageOS (OCaml), Clive (Golang), HalVM (Haskell), IncludeOS (C++), Ling (Erlang) and Runtime.js (Javascript)

Unikernel Essentials

- **The biggest characteristics**
  - Single address space: Zero-copy and huge page
  - Single running mode: Perform the efficient function call
  - One process with multiple threads: No heavy context switch and TLB flush
- **Compared to a traditional OS, unikernels provide many benefits**
  - Improved security
    - Unikernels reduce the amount of code deployed, which reduces the attack surface.
  - Small footprints
    - Unikernels images are often orders of magnitude smaller than traditional OS deployments.
  - Highly optimized
    - Unikernels enables whole-system optimization across device drivers and application logic. Especially it’s mostly paravirtualized under virtualization environment.
  - Fast Boot
    - Unikernels can boot extremely quickly, with boot times measured in milliseconds.

Unikernel Experiments: Public Claims

- **OSv**
  - For unmodified network-intensive applications, we demonstrate up to 25% increase in throughput and 47% decrease in latency. By using non-POSIX network APIs, we can further improve performance and demonstrate a 290% increase in Memcached throughput.
  - **http://www.cs.utah.edu/~peterm/prelim-osv-performance.pdf**
- **IncludeOS**
  - As a test case a bootable disk image consisting of a simple DNS server with OS included is shown to require only 158 kb of disk space and to require 5-20% less CPU-time, depending on hardware, compared to the same binary running on Linux.
- **ClickOS**
  - ClickOS virtual machines are small (5MB), boot quickly (about 30 milliseconds), add little delay (45 microseconds) and over one hundred of them can be concurrently run while saturating a 10Gb pipe on a commodity server.

Unikernel Challenges

- **Challenges - why existing unikernels have yet to gain large popularity**
  - Lack of compelling use cases
  - Compatibility with existing applications
  - Lack of production support (e.g. monitoring, debugging, logging)

Use Cases for Unikernels 1/3

- **Serverless**

  Most public cloud vendors are embracing this promising model with container.

  - Pros
    - Quick OS boot & improved security & smaller size and footprint
    - Mature VM management
    - Potentially multiple languages support
  - Cons
    - Unikernels is a little heavy to carry out just one function.
    - Debug issue can worsen serverless development.
    - Time of creating VM has a significant impact on function invocation.
  - Conclusion
    - In terms of different QOS unikernels are beneficial and useful complement to serverless mode. Furthermore, what if we can unikernelize linux, and further optimize it accordingly.

Use Cases for Unikernels 2/3

- **IoT**

  IoT is a big markets as well.

  – Pros
    - The feature of smaller size & footprint are good for those resource-strained IoT platforms.
    - Such a lightweight VM instance can address security issue.

  – Cons
    - Oftentimes unikernels need virtualization technology.
    - Unikernels are not designed to address those IoT characters like power consumption.
    - Unikernels don’t support versatile architectures.

  – Conclusion
    - Unikernels can value IoT when virtualization probably thrives at the edge. More importantly, IoT closely ties with the embedded system where Linux always plays a very import role, so it’s worth fitting unikernlized Linux into IoT.

Use Cases for Unikernels 3/3

- **IO intensive applications**

  IO Performance always captures people's attention.

  – Pros

    - Oftentimes unikernels have the simple IO flow framework

  – Cons

    - Only a subset of I/O intensive apps are good for unikernels: the latency-sensitive apps. The other subset of I/O intensive apps like the bandwidth-intensive apps need more considerations and explorations.

  – Conclusion

    - Unikernels can contribute IO case at large. NFV is really a potential chance to make unikernels succeed with any targeted acceleration to Linux.

Exploration Conclusions

- **Summary**
  - Unikernels still yield comparable performance.
    - The different unikernels have different focuses.
    - User has to put more or less effort to develop an application based on unikernels.
  - Nothing more specifically is done to embrace unikernels from hypervisor’s view.
  - Linux could be a good candidate of unikernels
    - Linux itself could help eliminate those challenges of unikernels
    - All optimizations and acceleration aimed to Linux can benefit unikernels
    - Unikernelized Linux can catch more eye by means of Linux community ☺

What Could We Do?

Our target is to explore what is the best platform for running unikernels case

We will achieve this by

- **Research existing unikernels**
  - Integrate and support those major existing unikernels well
- **Build new unikernel**
  - Convert Linux kernel
- **Explore optimizations**
  - Integrate virtIO model into ESXi as an example
  - Provide monitoring, logging and remote debugging
  - Supporting a short lived unikernels instance
  - Resources are consumed by live unikernels

What Are The Key Challenges?

- **Convert Linux to unikernels**
  - The fundamental philosophy of Linux is aiming to multiple processes and two modes.
  - Most components are coupled tightly.
  - How to further improve performance
- **Reduce time of creating VM**
  - Snapshort
  - VM Fork
- **A good paravirtualized API for common unikernels**
  - Some pv ops might already be a good start
- **New scheduler**
- **Manage the lifecycle and identities of the provisioned unikernels**

How Could We Possibly Achieve This? Hypervisor basics

- **Support major existing unikernels**
  - Integrate virtIO framework into ESXi
  - Port vmxnet3 and pvscsi into them
- **Define a standard API which can paravirtualize unikernels**
  - Based on common hypercall
  - Configure/control guest OS
  - Setup Inter-VM Communication
  - Allocate/destroy memory directly
- **Add a new scheduler**
  - Address short lived VM
  - Schedule a group of unikernels instances

How Could We Possibly Achieve This? Linux basics 1/2

- **Convert Linux**
  - Single running mode
    - Ring 0
      - __USER32_CS | __USER_DS | __USER_CS
      - Check with ‘cmpq $__USER_CS, CS(%rsp)’
    - Stack
      - Switch stacks manually
    - Interrupt Stack Table (IST)
      - set_intr_gate_ist(X86_TRAP_PF, &page_fault, PF_STACK)
      - Interrupt and exception
  - Single address space
    - Single process
    - No fork()/exec()

How Could We Possibly Achieve This? Linux basics 2/2

- **Convert Linux**
  - Optimization
    - Smaller size and footprint
    - Zero-copy
      - {get,put}_user
      - copy_{from,to}_user
      - Other unnecessary copy and check
    - Scheduler
      - scheduling classes & policies
        - fair vs rt vs deadline
        - New?
    - Lightweight TCP/IP Stack
      - LWIP
      - Fastsocket
      - Seastar
  - A variety of Linux variants
    - Multiple Unikernelized Linux profiles

How Could We Possibly Achieve This? Compatibility

- **Support existing applications**
  - Different code circumstances
    - Source code
      - New standard library
        - glibc
      - Function Call
    - Binary
      - –shared –pic
        - LD_PRELOAD
      - Others
        - BT
  - Multiple processes
    - One fork = one unikernelized Linux instance
      - IPC = Inter-VM Communication
    - PCID – Process-context identifiers
      - Limited bits
      - Linux's own debug/monitor/log tools and utilities

How Could We Possibly Achieve This?
Debugging, monitoring and logging

- **Debug unikernels**
  - Log info
    - virtual serial port
    - Dynamic buffer memory allocation
  - Linux's own utilizes
    - ssh/gdb/ftrace/perf/kprobe/kdump/…
    - PCID & the balloon driver
- **Monitor unikernels**
  - A mini-httpd as a stub connecting those Linux utilities
    - Inspired by OSv
- **Log unikernels**
  - rsyslog
  - vRealize Log Insight

How Could We Possibly Achieve This? Enhancements

- **Offer faster boot**
  - Explore ESXi to further reduce the time of creating VM
  - Skip BIOS with a small integrated bootloader
  - Replace ACPI with DTB
  - Adopt 1:1 Bus/device initialization
    - No any redundant bus scanning and device probing
- **Utilize hardware virtualization**
  - VT-X Instructions
    - VMFUNC
      - Pre-construct EPT table to get a faster and secure way to communicate between unikernels
  - VT-X Features
    - VPID (Virtual processor ID)
      - The tagged TLB to reduce cost of performance
    - Preempt Timer
      - A feature which count down in unikernels without too much external timer injected by hypervisor

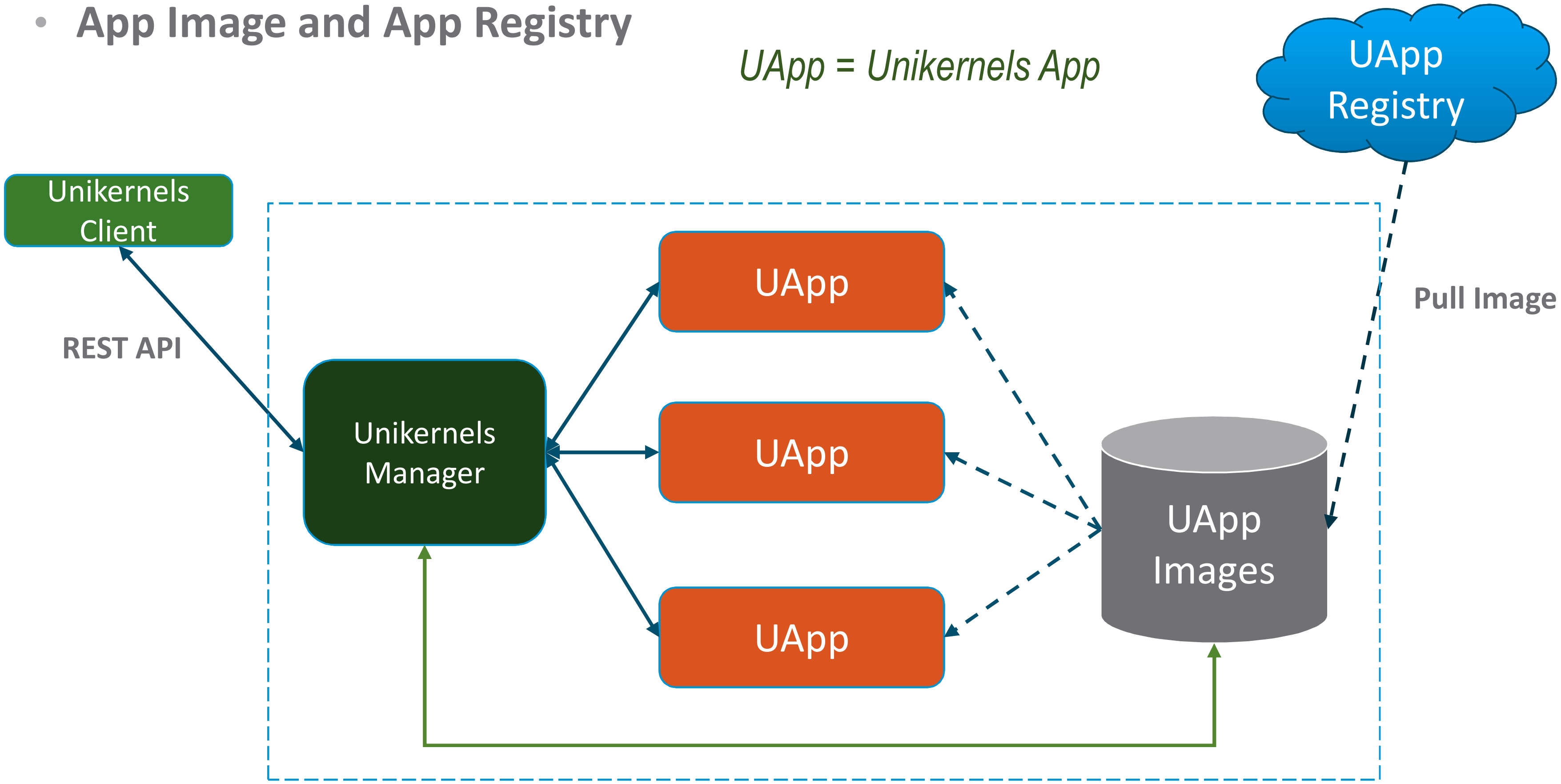
How Could We Possibly Achieve This? Others

- **Construct an efficient toolchain**
  - Build and deploy unikernels like Docker
  - Customized components management
    - Configuration
    - Kernel image
    - User App
    - Dependencies
- **Support orchestration**
  - Docker Swarm Mode, Kubernetes, Mesos and Cloud Foundry
  - **Unik**
- **Integrate Source Code Analyzer tool**
  - This can help us enhance security from code level

How Could We Possibly Achieve This? Management

- Unikernels Manager
- App Image and App Registry

UApp = Unikernels App



References

- http://unikernel.org/projects/
- https://wiki.xen.org/images/3/34/XenProject_Unikernel_Whitepaper_2015_FINAL.pdf
- https://www.linux.com/news/7-unikernel-projects-take-docker-2015
- https://www.usenix.org/node/184012
- https://www.deepdyve.com/lp/institute-of-electrical-and-electronics-engineers/includeos-a-minimal-resource-efficient-unikernel-for-cloud-services-J43NrzQ7fn
- https://www.usenix.org/system/files/conference/nsdi14/nsdi14-paper-martins.pdf

Thank You!

tiejunc@vmware.com

vmware