Unikernel and Explorations

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.
Agenda

- Unikernels Background
- Unikernels Explorations
  - Basic Solutions
  - What Are The Valuable Use Cases?
  - What Could We Do?
  - What Are The Key Challenges?
  - How Could We Possibly Achieve This?
Architectural Evolution Background

- VM with traditional OS
- Linux container technologies like Docker
- Container as a VM
- But now unikernels is really beginning to attract our attention!

Diagram:

(a) VM
(b) Container
(c) Container-in-VM
(d) Unikernel
Unikernels

• 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

• Unikernels still provide many comparable benefits
  – Improved security
  – Small footprints
  – Fast Boot
  – Highly optimized
Major Existing Unikernels

• **Unikernels approaches**
  - OSv, IncludeOS, MirageOS, ClickOS, Clive, HaLVM, LING, Rump Kernels, Solo5
    - Unikernel and Drawbridge
  - Unik

• **Unikernels solutions**
  - Docker
    - Hyperkit/VPNkit
    - Moby/Linuxkit
  - Mikelangelo
    - Improving Responsiveness and Agility of HPC Cloud Infrastructure
  - NFV
    - Unikernels based NFV architecture
Unikernel Experiments: Public Claims 1/2

• **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.”

• **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 Experiments: Public Claims 2/2

- Links
Research Conclusions

• **Status**
  – Unikernels really yield comparable performance.
  – 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)
  – Nothing more specifically is done to embrace unikernels from hypervisor’s view.
  – We probably need industry standard to Unikernels

• **Conclusion**
  – Linux could be a good candidate of unikernels
    • Linux itself could help eliminate those challenges
    • All optimizations and acceleration aimed to Linux can benefit unikernels
    • Fervent Linux community 😊
Use Cases for Unikernels 1/4

• IO intensive applications

  IO Performance always captures people’s attention.

  – Pros
    • Oftentimes unikernels have the simple and efficient 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.
Use Cases for Unikernels 2/4

- **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.
      
      - A group of functions
        - Serverless vs FaaS: Function as a Service
Use Cases for Unikernels 3/4

• IoT

IoT is one of the fastest-growing industries.

– 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 save, real-time requirement, etc.
  • 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 unikernelized Linux into IoT.
  • Unikernelized Linux can run on bare metal IoT Edge Gateway as well
Use Cases for Unikernels 4/4

• Others
  – Blockchain
    • Secure instance
  – Machine Learning
    • CPU/GPU support | Performance

• Conclusion
  • Unikernelized Linux is feasible technically in these cases.
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
  - Integrate virtIO model into ESXi as an example

- **Build new unikernel**
  - Convert Linux kernel

- **Explore optimizations**
  - 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  
    • Multiple processes  
    • Two modes  
  – Tightly coupled components  
  – How to further improve performance

• Reduce time of creating VM  
  – Snapshort  
  – VMware Instant Clone

• A good paravirtualized API for common unikernels  
  – Some existing pv ops

• 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 PV drivers 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 unikernels VM
  – Schedule a group of unikernels instances
How Could We Possibly Achieve This? Linux basics 1/3

- **Convert Linux**
  - Single Supervisor mode
    - Force setting Ring 0
      - `__USER32_CS | __USER_DS | __USER_CS`
      - `[GDT_ENTRY_DEFAULT_USER{32}_{CS:DS}]`
  - Using Interrupt Stack Table (IST)
    - `set_intr_gate_ist(X86_TRAP_PF, &page_fault, PF_STACK)`
    - Interrupt and exception
  - Rephrase VDSO
    - Redirect as function call
- **Manage Stack**
  - *Switch stack manually*
- **Single address space**
  - Single process with multiples threads
    - No fork()
How Could We Possibly Achieve This? Linux basics 2/3

• Convert Linux
  – Optimization
    • Smaller size and footprint
    • Zero-copy
      – access_ok
      – {get,put}_user
      – copy_{from,to}_user
      – Other unnecessary copy and check
        • clear_user/strnlen_user/strncpy_from_user
  • Scheduler
    – scheduling classes & policies
      • fair vs rt vs deadline
      • New?
  • Lightweight TCP/IP Stack
    – LWIP
    – Fastsocket
    – Seastar
    – …
How Could We Possibly Achieve This? Linux basics 3/3

• A variety of Linux variants
  – Standard Unikernelized Linux
  – Real-Time Unikernelized Linux
    • Preempt-RT Linux
  – Secure Unikernelized Linux
    • SELinux
    • Grsecurity Linux
  – Others
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 | ld.so.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/trace/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**
  – 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*
How Could We Possibly Achieve This? VDFS

• VDFS is a hyper-converged distributed file system with modern features

• Key features:
  – POSIX compliant
  – Support advanced features:
    – Distributed and scale-out
    – Zero config (no network to manage)
    – Multi-backend

• VDFS is an ideal platform for unikernels
  – Zero config
  – Shared file system cache
  – No need to manage disk images for unikernels
References

- http://unikernel.org/projects/
- https://www.usenix.org/node/184012
Thank You!

tiejunc@vmware.com