Extremely Fast and Efficient NFV with Unikraft

Sharan Santhanam
Felipe Huici

NEC Laboratories Europe GmbH

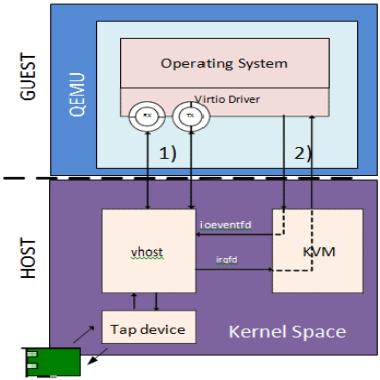
29th Oct 2020

UNICORE

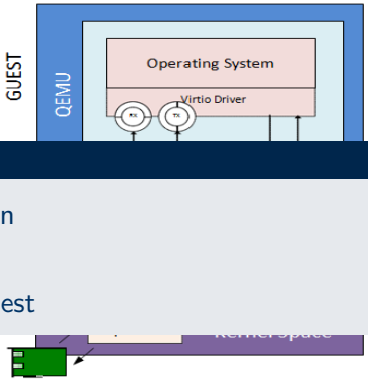
This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreements no.825377 ("UNICORE").This work reflects only the author's views and the European Commission is not responsible for any use that may be made of the information it contains.

Overview

VNF with DPDK Ecosystem



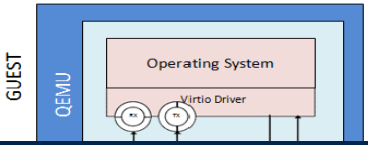
VNF with DPDK Ecosystem



Can we do better?

- ⇝ Guest OS specialization
- ⇝ Boot Time
- ⇝ Isolation within the guest

VNF with DPDK Ecosystem

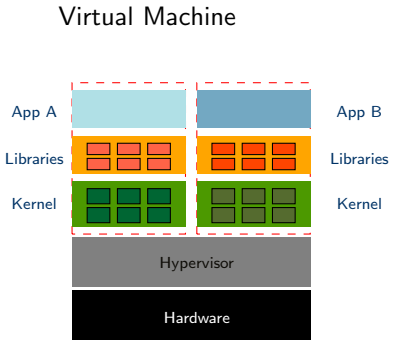


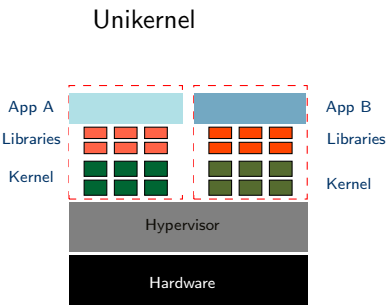
Can we do better?

- ⇝ Guest OS specialization
- ⇝ Boot Time
- ⇝ Isolation within the guest

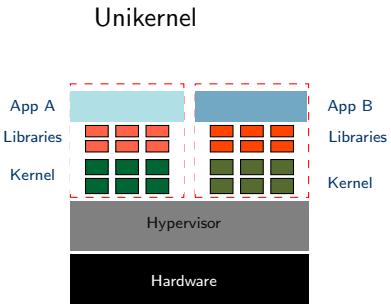
Let's discuss Unikernel...

Unikernel - Do One Thing and Do It Well



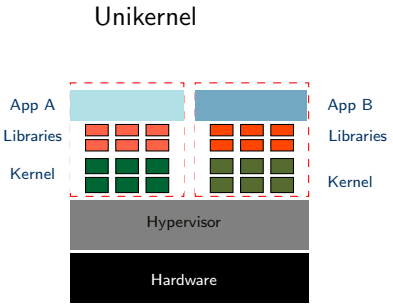


Unikernel - Do One Thing and Do It Well



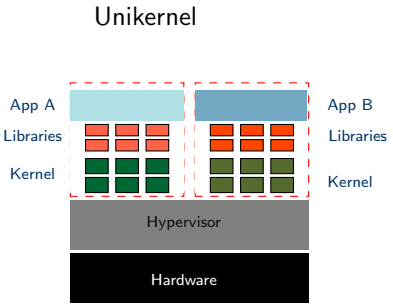
- Unikernel are purpose built
  - Thin kernel layer
  - Single image with application, specific kernel primitives

Unikernel - Do One Thing and Do It Well



- Unikernel are purpose built
  - Thin kernel layer
  - Single image with application, specific kernel primitives
- No isolation within a Unikernel
  - Flat address space

Unikernel - Do One Thing and Do It Well



- Unikernel are purpose built
  - Thin kernel layer
  - Single image with application, specific kernel primitives
- No isolation within a Unikernel
  - Flat address space
- Software Stack Specialization

"Really Unikernels!"

- Fast instantiation, destruction and migration times
  - 10 milliseconds or less
    (LightVM [Manco SOSP 2017], Jitsu [Madhvapeddy, NSDI 2015])

"Really Unikernels!"

- Fast instantiation, destruction and migration times
  - 10 milliseconds or less
    (LightVM [Manco SOSP 2017], Jitsu [Madhvapeddy, NSDI 2015])

- Low memory footprint
  - Few MBs of RAM or less
    (ClickOS [Martins NSDI 2014])

"Really Unikernels!"

- Fast instantiation, destruction and migration times
  - 10 milliseconds or less
    (LightVM [Manco SOSP 2017], Jitsu [Madhvapeddy, NSDI 2015])

- Low memory footprint
  - Few MBs of RAM or less
    (ClickOS [Martins NSDI 2014])

- High Deployment Density
  - 8k guests on a single x86 server
    (LightVM [Manco SOSP 2017])

"Really Unikernels!"

- Fast instantiation, destruction and migration times
  - 10 milliseconds or less
    (LightVM [Manco SOSP 2017], Jitsu [Madhvapeddy, NSDI 2015])

- Low memory footprint
  - Few MBs of RAM or less
    (ClickOS [Martins NSDI 2014])

- High Deployment Density
  - 8k guests on a single x86 server
    (LightVM [Manco SOSP 2017])

- High Performance
  - 10-40Gbit/s Ethernet throughput with a single guest CPU
    (ClickOS [Martins NSDI 2014], Elastic CDNs [Kuenzer VEE 2017])

"Really Unikernels!"

- Fast instantiation, destruction and migration times
  - 10 milliseconds or less
    (LightVM [Manco SOSP 2017], Jitsu [Madhvapeddy, NSDI 2015])

- Low memory footprint
  - Few MBs of RAM or less
    (ClickOS [Martins NSDI 2014])

- High Deployment Density
  - 8k guests on a single x86 server
    (LightVM [Manco SOSP 2017])

- High Performance
  - 10-40Gbit/s Ethernet throughput with a single guest CPU
    (ClickOS [Martins NSDI 2014], Elastic CDNs [Kuenzer VEE 2017])

- Reduced attack surface
  - Small trusted compute base
  - Strong isolation by hypervisor

La-la Land

So, Unikernel

- High Performance
- Isolation and reduced attack surface.
- Faster Instantiation Time
- Smaller image size

La-la Land

So, Unikernel

- High Performance
- Isolation and reduced attack surface.
- Faster Instantiation Time
- Smaller image size

The problem with Unikernel development:

- Building take several months or longer
- Potentially repeat the process for each target application
- **"Specialization" is hard to build**

La-la Land

So, Unikernel

- High Performance
- Isolation and reduced attack surface.
- Faster Instantiation Time
- Smaller image size

The problem with Unikernel development:

- Building take several months or longer
- Potentially repeat the process for each target application
- **"Specialization" is hard to build**

Ooops!!

That's not an effective way of doing things!

What is Unikraft?

Objectives

- ⇝ Support wide range of use cases
- ⇝ Simplify building and optimizing
- ⇝ Common and shared code base
- ⇝ Support different hypervisors
- ⇝ CPU architectures

What is Unikraft?

Objectives

- ⇝ Support wide range of use cases
- ⇝ Simplify building and optimizing
- ⇝ Common and shared code base
- ⇝ Support different hypervisors
- ⇝ CPU architectures

Unikraft

- ▶ "Everything is a library"
- ▶ Decomposed OS functionality
- ▶ Unikraft's two components:
  - Library Pool
  - Build Tool

What is Unikraft?

Objectives

- ⇝ Support wide range of use cases
- ⇝ Simplify building and optimizing
- ⇝ Common and shared code base
- ⇝ Support different hypervisors
- ⇝ CPU architectures

Unikraft

- ▶ "Everything is a library"
- ▶ Decomposed OS functionality
- ▶ Unikraft's two components:
  - Library Pool
  - Build Tool

Unikraft says Hi!!

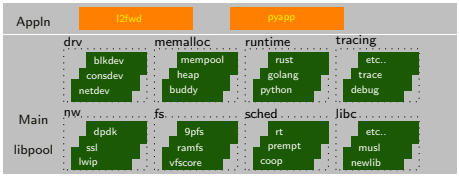
Source is BSD-licensed

Kconfig based build system

libukforest - Unikraft System Overview

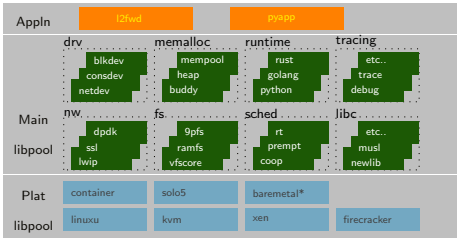
- Take an existing application
  - For example, a Python application or a l2fwd

libukforest - Unikraft System Overview



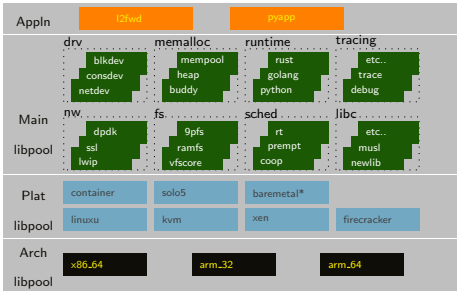
- Take an existing application
  - For example, a Python application or a l2fwd
- Pick Unikraft functionality
  - Pool of drivers and standard libraries

libukforest - Unikraft System Overview



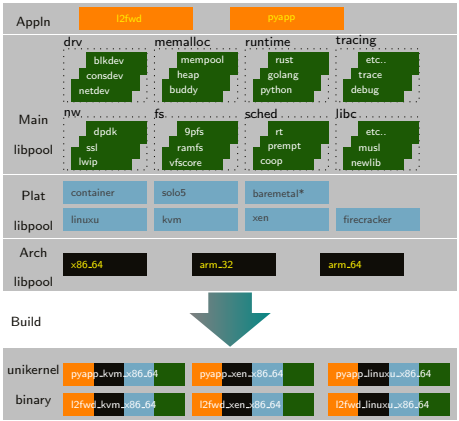
- Take an existing application
  - For example, a Python application or a l2fwd
- Pick Unikraft functionality
  - Pool of drivers and standard libraries
- Pick a platform and architecture
  - Pool of drivers and standard libraries

libukforest - Unikraft System Overview



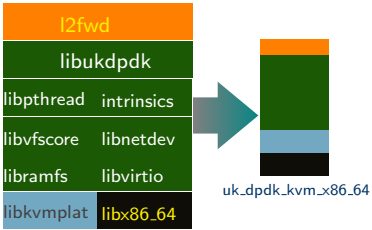
- Take an existing application
  - For example, a Python application or a l2fwd
- Pick Unikraft functionality
  - Pool of drivers and standard libraries
- Pick a platform and architecture
  - Pool of drivers and standard libraries
- Build Unikraft application

libukforest - Unikraft System Overview

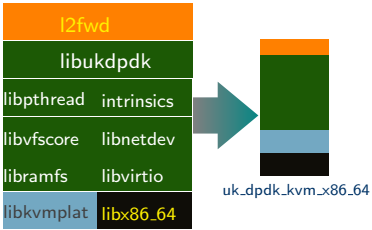


- Take an existing application
  - For example, a Python application or a l2fwd
- Pick Unikraft functionality
  - Pool of drivers and standard libraries
- Pick a platform and architecture
  - Pool of drivers and standard libraries
- Build Unikraft application

Unikraft - DPDK Target Arch?



Unikraft - DPDK Target Arch?



Challenges!!

- Build System Integration
- Specialization of Guest OS
- Minimize modification to DPDK library

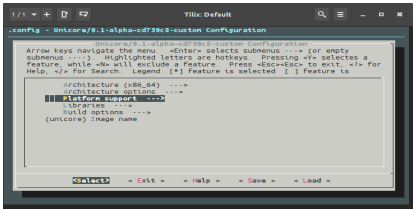
Build DPDK as an Unikraft Library

Unikraft Build system

- Config.uk (Kconfig based)
  - Handles dependencies across library
  - Enable/Disable Function

DPDK Build System

- Automatic config generation
  - CPU feature flags



Build DPDK as an Unikraft Library

Unikraft Build system

- Config.uk (Kconfig based)
  - Handles dependencies across library
  - Enable/Disable Function
- Makefile.uk (make based)
  - [LIBNAME]_SRCS
  - [LIBNAME]_CFLAG
  - CFLAG

DPDK Build System

- Automatic config generation
  - CPU feature flags
- Makefile (gmake)
  - SRCS
  - INCLUDE
  - CFLAG
  - DIRS

Build DPDK as an Unikraft Library

Unikraft Build system

- Config.uk (Kconfig based)
  - Handles dependencies across library
  - Enable/Disable Function
- Makefile.uk (make based)
  - [LIBNAME]_SRCS
  - [LIBNAME]_CFLAG
  - CFLAG
- exportsyms.uk

DPDK Build System

- Automatic config generation
  - CPU feature flags
- Makefile (gmake)
  - SRCS
  - INCLUDE
  - CFLAG
  - DIRS
- version map

Build DPDK as an Unikraft Library

Unikraft Build system

- Config.uk (Kconfig based)
  - Handles dependencies across library
  - Enable/Disable Function
- Makefile.uk (make based)
  - [LIBNAME]_SRCS
  - [LIBNAME]_CFLAG
  - CFLAG
- exportsyms.uk

DPDK Build System

- Automatic config generation
  - CPU feature flags
- Makefile (gmake)
  - SRCS
  - INCLUDE
  - CFLAG
  - DIRS
- version map

libukdpdkbuild

- Process DPDK Makefile.

Build DPDK as an Unikraft Library

Unikraft Build system

- Config.uk (Kconfig based)
  - Handles dependencies across library
  - Enable/Disable Function
- Makefile.uk (make based)
  - [LIBNAME]_SRCS
  - [LIBNAME]_CFLAG
  - CFLAG
- exportsyms.uk

DPDK Build System

- Automatic config generation
  - CPU feature flags
- Makefile (gmake)
  - SRCS
  - INCLUDE
  - CFLAG
  - DIRS
- version map

libukdpdkbuild

- Process DPDK Makefile.
  - Add DPDK library

Build DPDK as an Unikraft Library

Unikraft Build system

- Config.uk (Kconfig based)
  - Handles dependencies across library
  - Enable/Disable Function
- Makefile.uk (make based)
  - [LIBNAME]_SRCS
  - [LIBNAME]_CFLAG
  - CFLAG
- exportsyms.uk

DPDK Build System

- Automatic config generation
  - CPU feature flags
- Makefile (gmake)
  - SRCS
  - INCLUDE
  - CFLAG
  - DIRS
- version map

libukdpdkbuild

- Process DPDK Makefile.
  - Add DPDK library
  - Support newer version of DPDK

Build DPDK as an Unikraft Library

Unikraft Build system

- Config.uk (Kconfig based)
  - Handles dependencies across library
  - Enable/Disable Function
- Makefile.uk (make based)
  - [LIBNAME]_SRCS
  - [LIBNAME]_CFLAG
  - CFLAG
- exportsyms.uk

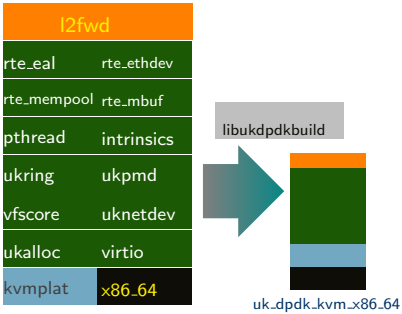
DPDK Build System

- Automatic config generation
  - CPU feature flags
- Makefile (gmake)
  - SRCS
  - INCLUDE
  - CFLAG
  - DIRS
- version map

libukdpdkbuild

- Process DPDK Makefile.
  - Add DPDK library
  - Support newer version of DPDK
- Add dpdk specific configuration file.

Unikraft - DPDK Target Arch



Unikraft as EAL in DPDK

Specialize the Guest OS

- Memory management
- Bus/Device Management
- CPU Scheduling and CPU Features

Unikraft as EAL in DPDK

Specialize the Guest OS

- Memory management
- Bus/Device Management
- CPU Scheduling and CPU Features

Memory Management

- Unikraft: flat page table since boot
- Huge pages based 2MB sized pages
- Memory region can be explicitly assigned to the Application
- Custom memory allocator per memory region

Unikraft as EAL in DPDK

Specialize the Guest OS

- Memory management
- Bus/Device Management
- CPU Scheduling and CPU Features

Bus/Device Management

- A simpler bus/device interface
- Directly attached device and usable by DPDK with unikraft

Unikraft as EAL in DPDK

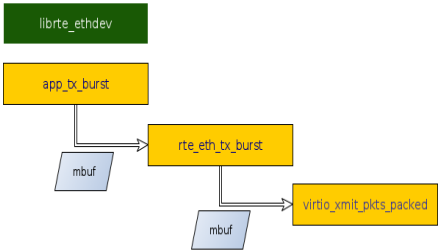
Specialize the Guest OS

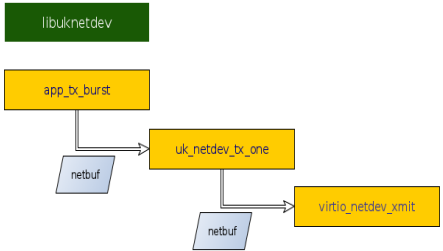
- Memory management
- Bus/Device Management
- CPU Scheduling and CPU Features

CPU Scheduling and CPU Features

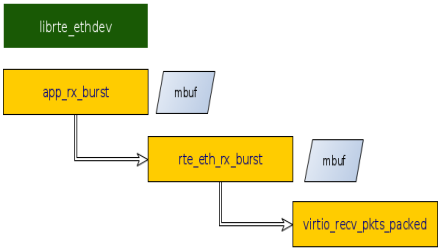
- Application decides on scheduling on the core.
- Minimal interference / resource usage for other purpose within guest.

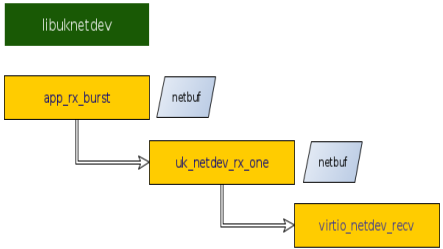
Interface between Unikraft and DPDK



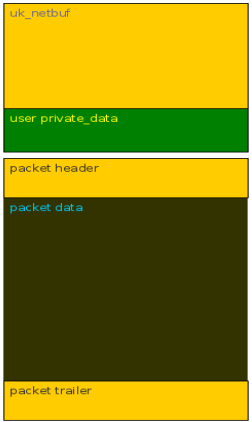


Interface between Unikraft and DPDK

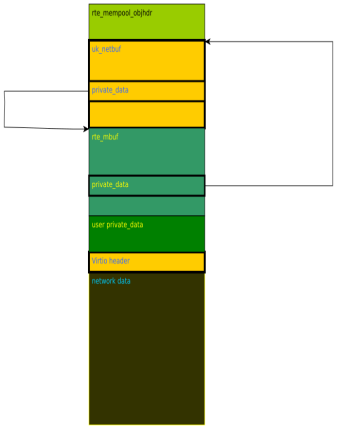




Interface between Unikraft and DPDK



Interface between Unikraft and DPDK



Test Setup

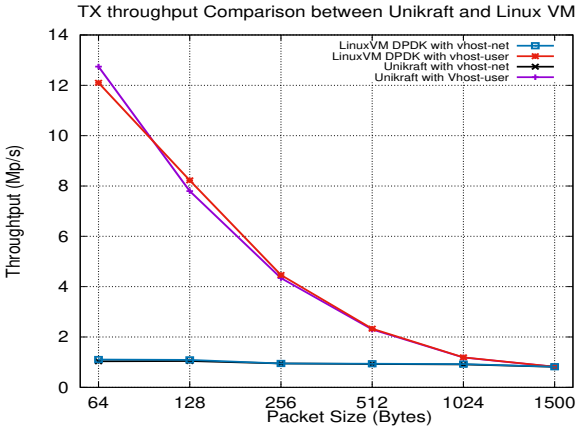
Unikraft/Linux GuestVM Setup

- CPU Family: Sandy Bridge (Server) (Family: 6 and Model: 45)
- CPU Model: Intel(R) Xeon(R) CPU E5-1650 0 @ 3.20GHz
- Nr of Cores: 6
- RAM : 16GB
- Nr of NUMA nodes: 1
- Host Linux Kernel: Debian 4.19
- Guest Linux Kernel: Debian 4.19
- Qemu Version: 4.0.0
- DPDK Version (vhost-user): 19.08

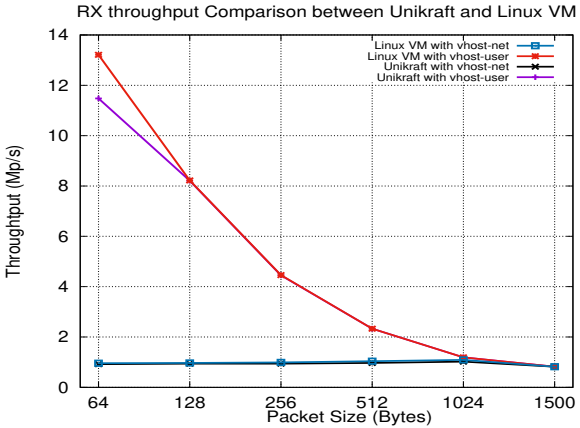
Packet Generator/Receiver Setup

- CPU Family: Ivy Bridge (Server) (Family: 6 and Model: 62)
- CPU Model: Intel(R) Xeon(R) CPU E5-1620 v2 @ 3.70GHz
- Nr of Cores: 4
- RAM : 16GB
- Nr of NUMA nodes: 1
- Host Linux Kernel: Debian 4.19
- DPDK Version (vhost-user): 19.08

Packet Send: Unikraft vs LinuxVM



Packet Receive: Unikraft vs LinuxVM



Key Value: Unikraft vs LinuxVM

Linux Guest VM	Unikraft using DPDK(mbuf)	Unikraft using Netdev(netbuf)
6.1Mp/s	6.1Mp/s	6.0Mp/s

Resource Usage: Unikraft vs LinuxVM

Parameters	Unikraft	Linux VM
Memory	1GB	6GB
Boot Times	87ms	12s
Image Size	1.4MB	2.5GB

Future Work

- Add SMP support
- Add NUMA support
- Use DPDK drivers directly

Join us!

Project Home Page

http://unikraft.org

Documentation

http://docs.unikraft.org/

Sources

https://github.com/unikraft

http://xenbits.xen.org/gitweb/ (Namespace: Unikraft)

What we think

Unikraft

- Support multiple platforms
- Specialized Guest OS
- Simpler Management Device

- Increased control for an application

DPDK

- Performance of Network stack
- Specialized VNF
- Wealth of knowledge DPDK driver

- Increased application base

What we think

Unikraft

- Support multiple platforms
- Specialized Guest OS
- Simpler Management Device
- Increased control for an application

DPDK

- Performance of Network stack
- Specialized VNF
- Wealth of knowledge DPDK driver
- Increased application base

What do you think?