Unikernels: Library Operating Systems for the Cloud

Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand and Jon Crowcroft

ASPLOS'13

In The Context of the CS3551 Spire Class Project

Brad Whitehead and Mike Boby

February 25, 2020

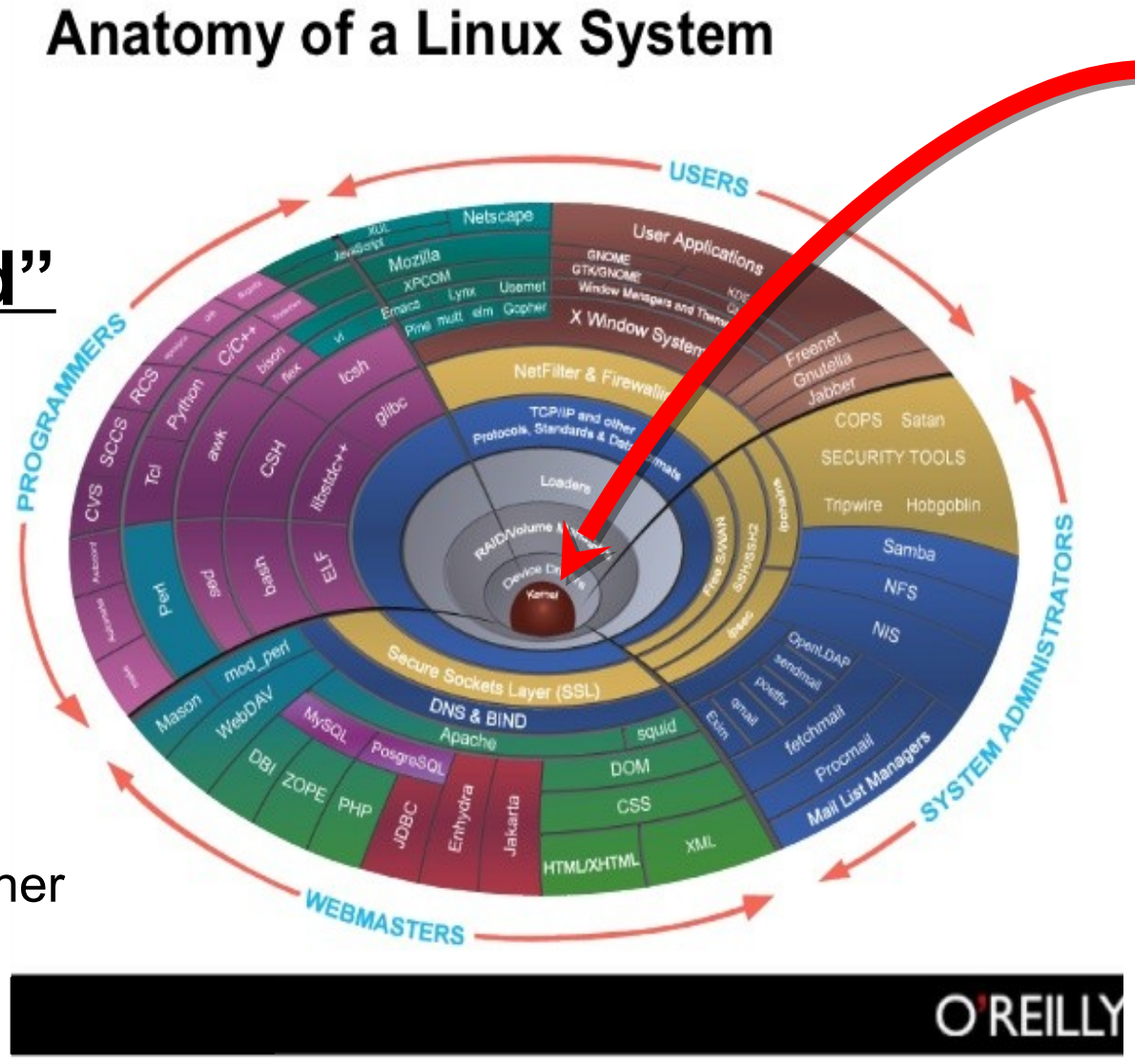
Teaching Moment - What is a Unikernel?

- To answer that question, we have to take a look at the structure of a modern operating system
- Doesn't matter if it's Microsoft Windows, Linux, UNIX, ~~Mac OS X~~ ~~OS X~~ macOS, etc.
- All the mainstream operating systems have the same fundamental anatomy

The Anatomy of an Operating System

“Userland”

- Where Applications Run
- No Privileges
- Can Not Access Resources
-
- Can Not “Talk” to Other Programs



“The Kernel”

- Runs in Special Hardware Mode
  - – “Ring 0”
- Only Program That Can Access or Allocate Resources
- Copies Data Between Programs

Growth of Operating Systems

- Linux Kernel is now 28 million lines of source code!
- Windows is estimated at 50 million lines of code!!
- With an industry average of 15-50 defects per 1000 lines of code*:
  - Linux (Just the kernel) = 420 thousand to 1.4 million defects
  - Windows = 750 thousand to 2.5 million defects
  - * Steve McConnel, “Code Complete 2”, 2005

It Gets Worse

- The “Userland” support software is often 10 to 20 times larger than the kernel
- Red Hat Enterprise Linux (RHEL) Userland is approximately 420 million lines of code
  - Try not to think about the 6.7 to 22 million defects running on the SCADA server controlling your power grid and drinking water

Can It Get Even Worse?

- The Linux kernel is full of junk!
- A large number of device drivers are routinely compiled into the kernel, regardless of the actual hardware in the computer
  - There are device drivers for hardware that no longer exists
    - Silicon Graphics video drivers were just added to the Linux 5.5 kernel!
  - Amazon AMI images ~~have~~ had drivers for floppy disks and audio cards!
    - In 2015, the Venom vulnerability (CVE-2015-3456) used a flaw in the floppy disk controller (FDC) driver to compromise both physical and virtual machines

And It Doesn't End There

- Likewise, there are thousands of storage and communications protocols in the kernel that will not be used in your application
- Linux recognizes 7 different executable formats, even though the vast majority of applications (including Spire) are in ELF format
- Each of these extra, unused chunks of code (with its 15-50 defects/1000 SLOC) is a potential entry point for compromise

What If We Cut Out the Parts We Don't Need?

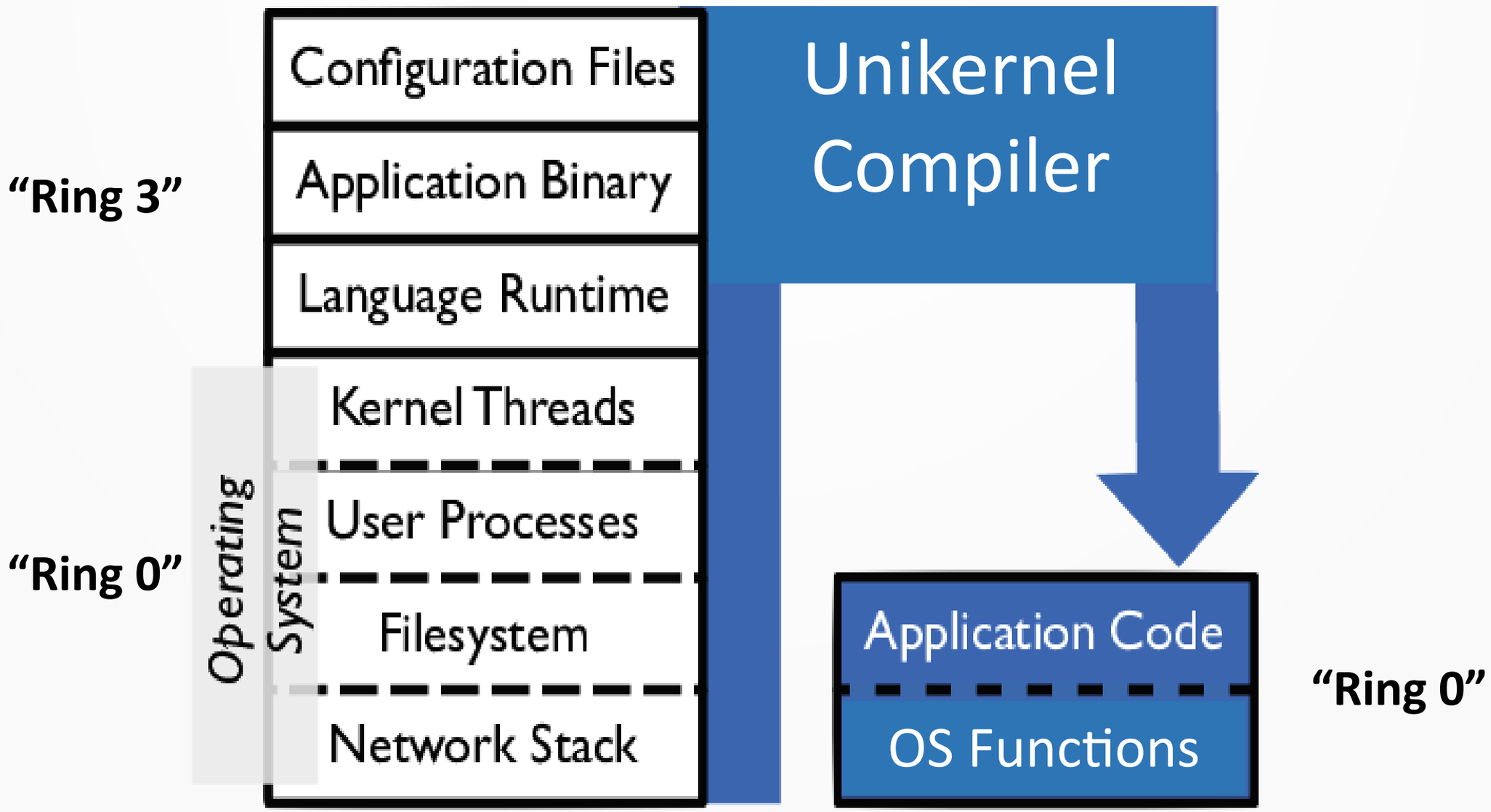
- Code traces show that the average application uses less than 0.08% of the total code in the kernel
- Take the standard C library as an example
  - The C library contains thousands of functions, but a modern linker only includes the actual functions (and code) that an application uses
- Could we do the same with our operating system?

We Can Create A Library of Operating System Functions

- Common operating system functions, drivers, and protocols are written as a library of functions
- When you link these “library operating system” functions to your application, you have a single executable that runs directly on hardware or a hypervisor as a stand-alone virtual machine (VM)…
  - Only the functions, drivers, and protocols actually used in the application are linked into the executable/VM

…You have a Unikernel!

So What Does a Unikernel Look Like?



Unikernel Approaches

- "From the Ground Up" - Programmer writes code specifically for the Unikernel library functions
  - Mirage
- "Partition an Existing Kernel" - Extract all the functions of an existing kernel and include only needed functions during compile
  - Rump kernel
- "POSIX/Linux Interface to New Library Functions" - New, modernized functions are written for the library, but clients call the functions through existing interfaces
  - IncludeOS

The Paper's Focus

- Mirage
  - A set of libraries that perform the functions commonly associated with the operating system for memory management, execution, and communications
  - Written in a strongly typed functional language, OCaml
  - OCaml applications linked with the Mirage libraries form virtual machine images (unikernels) designed to be run on the Xen hypervisor
  - Mirage unikernels use Xen for device drivers and scheduling

The Paper - Why Unikernels?

- Enhanced Security
  - Hardware-Enforced Access Controls
  - Less Code
  - Immutable
  - No System Calls
  - Per-Compile Randomization
- Single Address Space
  - No Expensive Context Switches
  - Zero Copy
- Small Executable Size
- Reduced Runtime Complexity (No Scheduler)

Architecture of a Unikernel (1/3)

- Configuration and Deployment
  - Instead of /etc and config files, the application configuration is defined at compile time and compiled directly into the executable code.
  - Configurations are explicit and manipulated directly by the high level language making them subject to type checking and static anaysis
  - Reduced effort to configure multi-service applications
- Compactness and Optimization
  - Using only the required functions makes for compact code
  - Since the compiler sees all the code, it can apply optimizations to the entire unikernel

Architecture of a Unikernel (2/3)

- Threat Model
  - Tenants in a shared cloud environment (possibly Spire data center controllers)
  - Across the network (definitely Spire)
  - Hypervisor provides isolation and access control
  - Compile time specialization (no system calls, no scheduler, etc.)
- Single Image
  - Removal of all unused functions and code
- Pervasive Type Safety
  - Mirage uses a single, strongly typed language

Architecture of a Unikernel (3/3)

- Sealing and Privilege Dropping
  - Mark code as immutable, enforced by hypervisor
    - Code pages are marked “read-only”
    - Data pages (stack, heap, mmap, etc.) are marked “non-executable”
  - Harvard architecture instead of Von Neumann architecture
- Compile-Time Address Space Randomization
  - Mirage unikernel toolchain can produce randomized internal addresses (equivalent to ASLR)

Components of the Mirage Unikernel (1/4)

- OCaml
  - The majority of the operating system functions are written from scratch in OCaml, a strongly typed functional language
  - The authors attribute much of Mirage's reliability to the use of OCaml
- PVBoot Library
  - Minimal code required to:
    - Create a single 64 bit address space
    - Load unikernel image
    - Allocate required memory to unikernel data structures
    - Use 1 vCPU
    - Connect to Xen event channels
  - Compiled directly into the unikernel image

Components of the Mirage Unikernel (2/4)

- Language Runtime
  - Mirage uses a specialized OCaml runtime library
    - Modified for single address space layout
    - Memory mapped I/O between Mirage unikernel VMs on the same Xen hypervisor
  - PVBoot provides a single event-driven execution loop
  - Thread concurrency comes from a Lightweight Thread Library written in OCaml
- Device Drivers
  - Mirage uses Xen device drivers
  - Xen device drivers communicate with VMs using a single shared memory page of “slots” arranged in a ring buffer, with event channels for signaling
  - Mirage wraps this Xen ring I/O within OCaml functions for type safety enforcement

Components of the Mirage Unikernel (3/4)

- Zero-Copy Device I/O
  - With a single address space, no need to copy data from kernel space to user space
- Type-Safety Protocol I/O
  - All I/O is wrapped in OCaml for type safety, eliminating buffer overflow errors/attacks

Components of the Mirage Unikernel (4/4)

- Network Processing
  - Fast shared memory between unikernels in the same hypervisor
  - “Scatter/Gather” approach to build and send Ethernet TCP/IP I/O
- Storage
  - Uses “shared page” I/O ring buffer with Xen hypervisor for block storage
  - OCaml library in unikernel provides filesystem abstraction over the blocks

Evaluation (1/3)

- Microbenchmarks
  - Boot Time
    - Mirage boots in 50 milliseconds, versus 500 milliseconds for an equivalent Linux VM
  - Threading
    - Mirage can launch 20 million threads per second, versus 4 seconds for an equivalent Linux VM
  - Networking and Storage
    - Mirage was 4-10% slower than Linux VM when processing ICMP Ping requests
    - Mirage was slightly faster than Linux on IPv4 reads (zero-copy) and slightly slower on writes because of CPU operations in protocol libraries
    - Mirage and Linux direct I/O storage throughput effectively the same (1.6 GB/sec)

Evaluation (2/3)

- DNS Server Appliance (Unikernel image – 183.5 kB versus Linux image - 462MB)
  - BIND9 – 55K queries per second
  - NSD – 70K queries per second
  - Mirage DNS appliance – 75-80K queries per second
- OpenFlow Controller Appliance

Program	Batch	Single Request
Maestro	20K	10K
NOX	120K	40K
Mirage Appliance	100K	35K

Evaluation (3/3)

- Dynamic Web Server Appliance
  - Twitter application
    - Mirage appliance – 800 sessions per second
    - Linux – 200 sessions per second
  - Static web server
    - Mirage appliance ~ 2000 connections per second
    - Apache2 ~ 1700 connections per second
- Code and Binary Size
  - Mirage unikernels are 4-5 X fewer SLOC than an equivalent Linux appliance (after maximum stripping of Linux)

Paper Conclusions

- Demonstrated that the unikernel approach significantly improves safety and efficiency for cloud appliances
- Contributed a “clean slate” library OS based on the strong typed OCaml functional language
- Performance equal to, and in some cases better than, conventional operating system-hosted applications
- By relaxing (abandoning) backwards compatibility, safety and efficiency can be improved

Differences Between Mirage and Our Spire Project

- Mirage Assumes New Code Development - “From the Ground Up”
- We Will Be Using a Unikernel Library Designed to Support Existing POSIX/Linux Software
- While Mirage Can Randomize Internal Addresses, Most Unikernel Libraries Can Not
- Combine Unikernel Approach With MultiCompiler Approach
  - “Do Nothing ;-)”

What Do We Need for Spire?

- Run a single application per server (no scheduler required)
- Run as a single user
- Uses a known set of hardware drivers
- Uses 1 or 2 communications protocols
- Needs security (from unauthorized access - “hacking”)
- Needs reliability
- Nice to-have: Speed (low startup and processing latency)

Keeping Only The OS Functions That Spire Requires

- What does that buy us?
- Let's start with security:
  - Greatly reduced attack surface (99.92% reduction)
- We don't need any userland applications (bye-bye 410 million lines of potentially flawed code!)
  - No shell (/bin/sh)
- No ability to run malicious or hacking tools on the same VM
- Function calls instead of system calls (more secure)
- No time consuming context switches
- Static linking with prevent injection attacks
- No re-configuration attacks

How To Include Only Required Code?

- The C Library Analogy Is The Key
- The C Library Is Actually A “Middle Ware Layer”
  - It Converts Standard C Function Calls Into Equivalent Kernel System Calls
  - Instead of Handing the Function Call Off as a System Call, What If We Extended the C Library to Include the Appropriate Kernel Code?
  - Instead of the C Library Passing a “Print()” Call To The Kernel, the Library Can Include the Machine Instructions to Do The Actual I/O

Increased Performance

- Smaller, Less Memory Intensive Images Mean More Virtual Machines Per Hardware Server
  - 5 Megabyte Virtual Machines = 10,000 VMs Per Physical Server
  - Smaller Than Most Docker Containers
- 6 Millisecond Boot Times
  - Jitsu – Boot-On-Demand
- 45 Microsecond Throughput Times
  - No Context Switches
  - No Information Copying
  - Single Address Space