MANCHESTER
1824

The University of Manchester

Compatibility and Isolation in Specialised Operating Systems

Pierre Olivier

The University of Manchester
pierre.olivier@manchester.ac.uk

17/05/2023

Table of contents

Introducing Unikernels

HermiTux: a Unikernel Binary-Compatible with Linux

FlexOS: an Operating System for Flexible Isolation

Table of contents

Introducing Unikernels

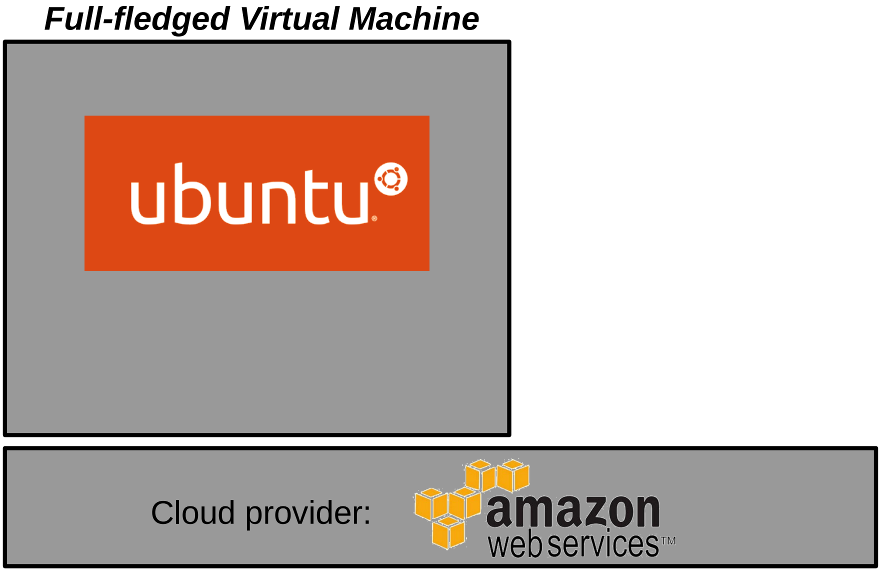
HermiTux: a Unikernel Binary-Compatible with Linux

FlexOS: an Operating System for Flexible Isolation

Introductory example:
my website in the cloud

Introducing Unikernels

Presentation

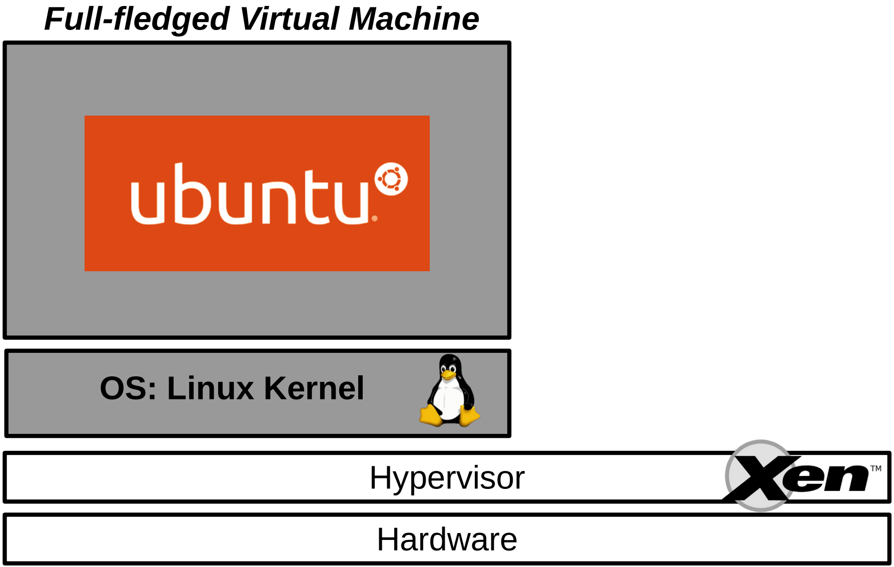


Full-fledged Virtual Machine

Cloud provider:

Introducing Unikernels

Presentation

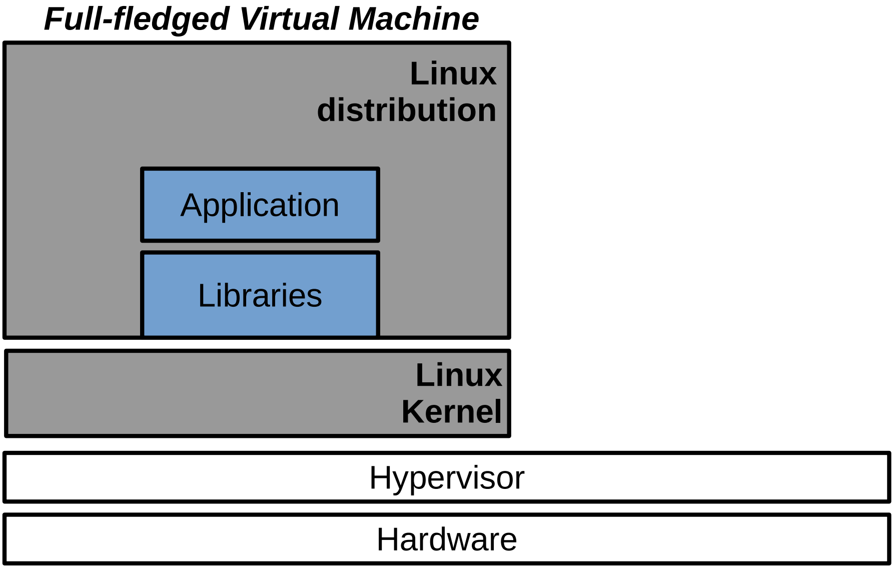


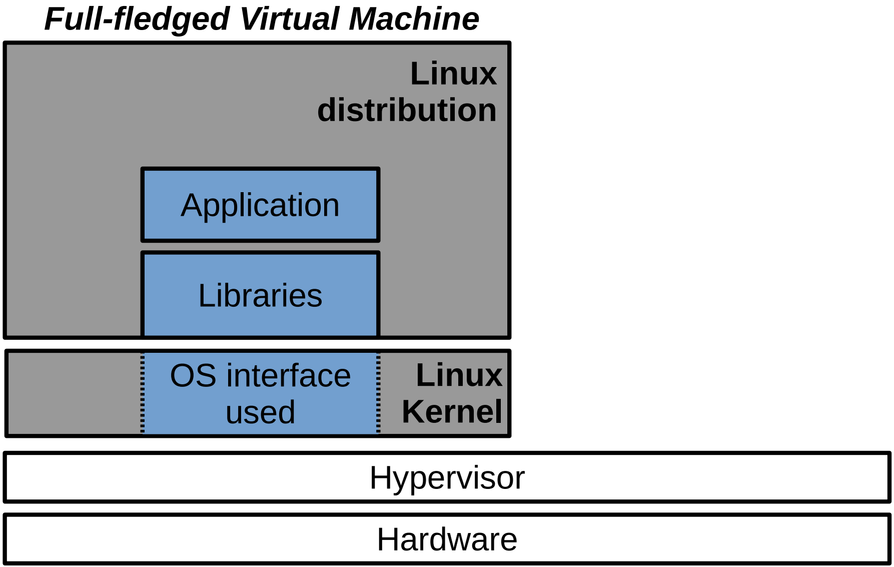
Full-fledged Virtual Machine

OS: Linux Kernel

Hypervisor

Hardware

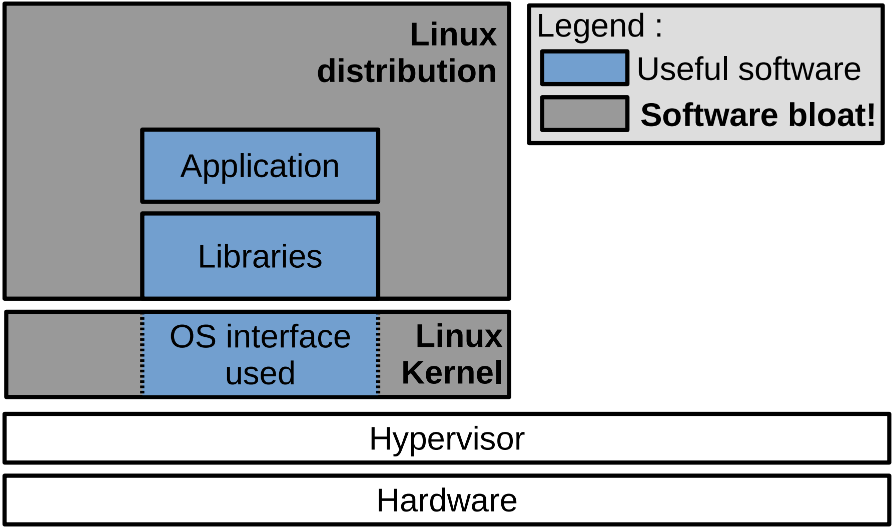

Full-fledged Virtual Machine
Linux distribution
Application
Libraries
Linux Kernel
Hypervisor
Hardware


Full-fledged Virtual Machine
Linux distribution
Application
Libraries
OS interface used
Linux Kernel
Hypervisor
Hardware

Introducing Unikernels

Presentation

Full-fledged Virtual Machine



Linux distribution

Application

Libraries

OS interface used

Linux Kernel

Hypervisor

Hardware

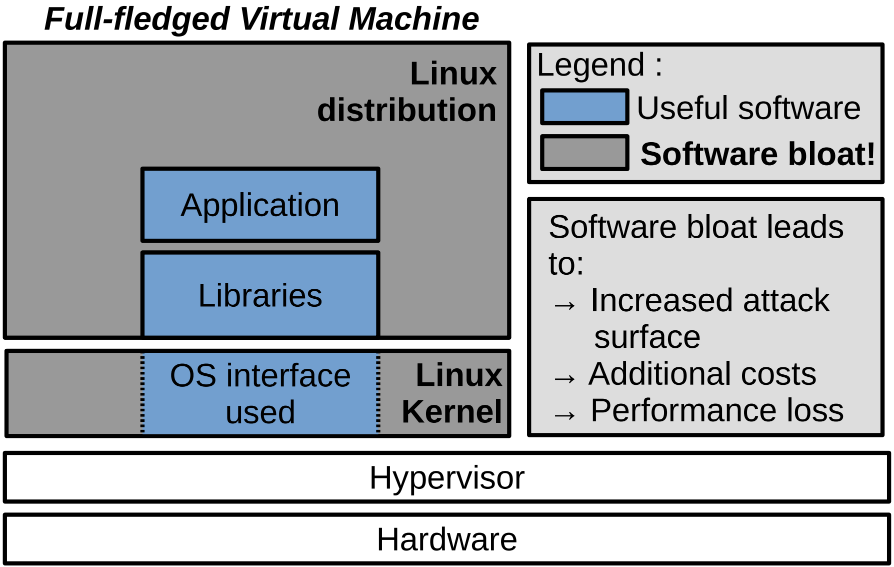
Legend :

🟦 Useful software

⬜ **Software bloat!**

Introducing Unikernels

Presentation



Full-fledged Virtual Machine

Linux distribution

Application

Libraries

OS interface used

Linux Kernel

Hypervisor

Hardware

Legend :

🟦 Useful software

▨ **Software bloat!**

Software bloat leads to:

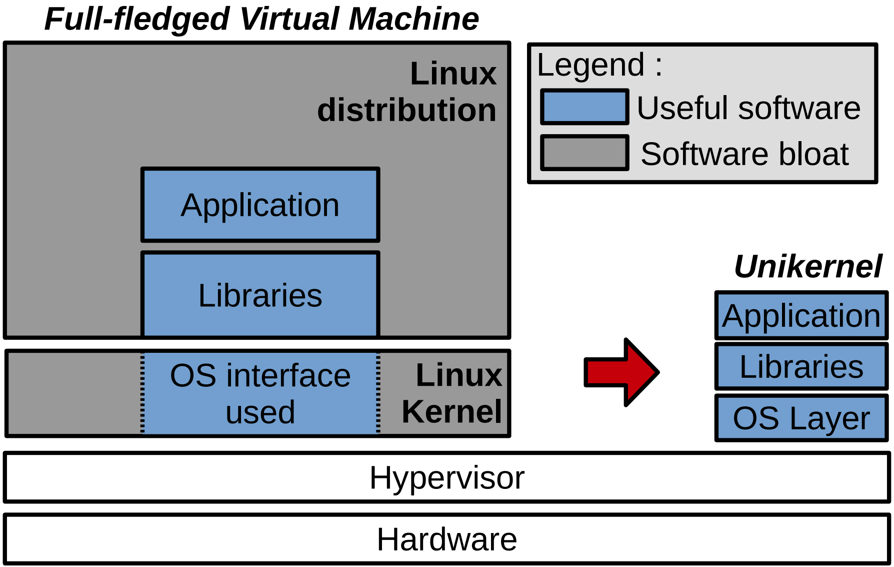
→ Increased attack surface

→ Additional costs

→ Performance loss

Introducing Unikernels

Presentation



Introducing Unikernels

Presentation (2)

Unikernel: application + dependencies + thin OS compiled as a static binary running on top of a hypervisor [1]

[1] Madhavapeddy et al., “Unikernels: Library Operating Systems for the Cloud”, ASPLOS’13

[2] Zhang et al., “KylinX: A Dynamic Library Operating System for Simplified and Efficient Cloud Virtualization, ATC’18

Introducing Unikernels

Presentation (2)

Unikernel: application + dependencies + thin OS compiled as a static binary running on top of a hypervisor [1]

- **Single-purpose**: run 1 application
  - Want to run multiple applications? Run multiple unikernels
- **Single-process**
  - Want to run a multi-process application? Run multiple unikernels [2]
  - However, SMP (multicores) and multithreading are supported
- **Single-binary and single-address space** for application + kernel
  - No user/kernel protection needed

[1] Madhavapeddy et al., “Unikernels: Library Operating Systems for the Cloud”, ASPLOS’13

[2] Zhang et al., “KylinX: A Dynamic Library Operating System for Simplified and Efficient Cloud Virtualization, ATC’18

Introducing Unikernels

Presentation (3)

- **Lightweight virtualisation**
    - Contain and run only what is absolutely necessary for the application
    - Security advantage: small attack surface
    - Cost advantage: memory/disk footprint reduction
    - As VMs, considered as a *secure alternative to containers*

Introducing Unikernels

Presentation (3)

- **Lightweight virtualisation**
  - Contain and run only what is absolutely necessary for the application
  - Security advantage: small attack surface
  - Cost advantage: memory/disk footprint reduction
  - As VMs, considered as a *secure alternative to containers*
- **Per-application tailored kernel**
  - LibOS/Exokernel model, specialised subsystems
  - The kernel itself contains only what is needed

Introducing Unikernels

Presentation (3)

- **Lightweight virtualisation**
  - Contain and run only what is absolutely necessary for the application
  - Security advantage: small attack surface
  - Cost advantage: memory/disk footprint reduction
  - As VMs, considered as a *secure alternative to containers*
- **Per-application tailored kernel**
  - LibOS/Exokernel model, specialised subsystems
  - The kernel itself contains only what is needed
- **Reduced OS noise, increased performance**
  - Low system call latency: app + kernel in ring 0, system calls are function calls
  - Sub-second boot time

Introducing Unikernels

Unikernels Models

Unikernels can be classified based on the language(s) in which the applications they support can be written in (non-exhaustive list):

- Pure **memory safe languages** (OCamL, Erlang, Haskell): MirageOS[1], LING[2], HalVM[3]
- **C/C++, semi-POSIX API**: HermitCore[4], OSv[5], Rumprun[6], Lupine[7], UKL[8]
- **Rust/Go**: RustyHermit[9], Clive[10]
- **Multi-languages**: HermiTux[11], Unikraft[12]

[1] https://mirage.io/, [2] https://github.com/cloudozer/ling [3] https://github.com/GaloisInc/HaLVM [4] https://hermitcore.org/ [5] http://osv.io/ [6] https://github.com/rumpkernel/rumprun [7] https://github.com/hckuo/Lupine-Linux [8] https://github.com/unikernelLinux/ukl [9] https://github.com/hermitcore/libhermit-rs [10] http://lsub.org/ls/clive.html [11] https://ssrg-vt.github.io/hermitux/ [12] https://unikraft.org/

Introducing Unikernels

Summary

Benefits Summary

- Lower costs by being lightweight
- Increased security with low attack surface and high isolation
- Increased performance with low OS disturbance and fast boot times

Introducing Unikernels

Summary

Benefits Summary

- Lower costs by being lightweight
- Increased security with low attack surface and high isolation
- Increased performance with low OS disturbance and fast boot times

Application domains

- Cloud applications: servers, micro-services, SaaS, FaaS/serverless
- Embedded virtualisation, Edge computing, IoT
- Network Function Virtualisation, HPC, VM introspection, malware analysis, secure Desktop applications
- etc.

Table of contents

Introducing Unikernels

HermiTux: a Unikernel Binary-Compatible with Linux

FlexOS: an Operating System for Flexible Isolation

HermiTux: a Unikernel Binary-Compatible with Linux

The Issue

- Unikernels have plenty of benefits to bring
- Unikernels have plenty of application domains
- They are very popular in academia ...
  - Mirage (ASPLOS’13), LightVM (SOSP’17), and many others
- **...but why (nearly) nobody uses them in the industry?**

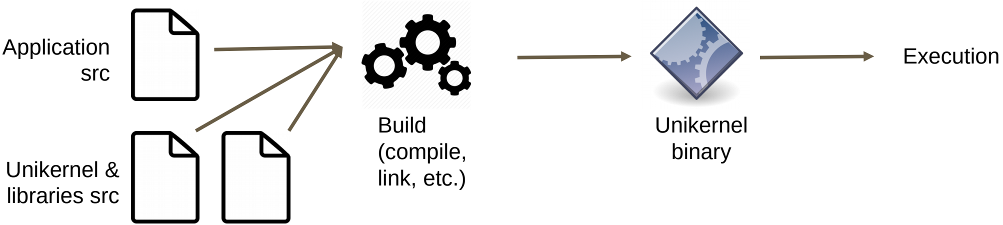
The Issue

- Unikernels have plenty of benefits to bring
- Unikernels have plenty of application domains
- They are very popular in academia …
  - Mirage (ASPLOS'13), LightVM (SOSP'17), and many others
- **…but why (nearly) nobody uses them in the industry?**

Because it is hard to port existing applications!

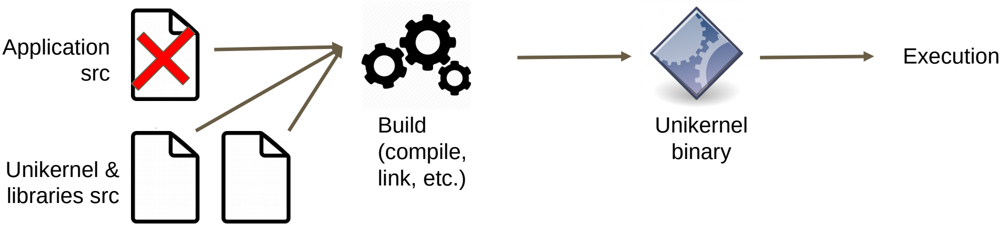
HermiTux: a Unikernel Binary-Compatible with Linux

Why Porting to Unikernels is Hard



HermiTux: a Unikernel Binary-Compatible with Linux

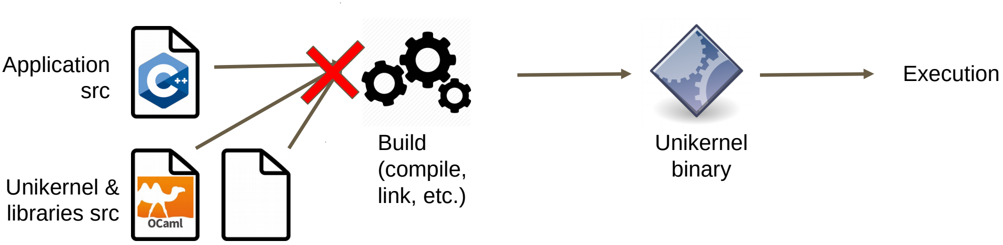
Why Porting to Unikernels is Hard



- Proprietary software → source code not available

HermiTux: a Unikernel Binary-Compatible with Linux

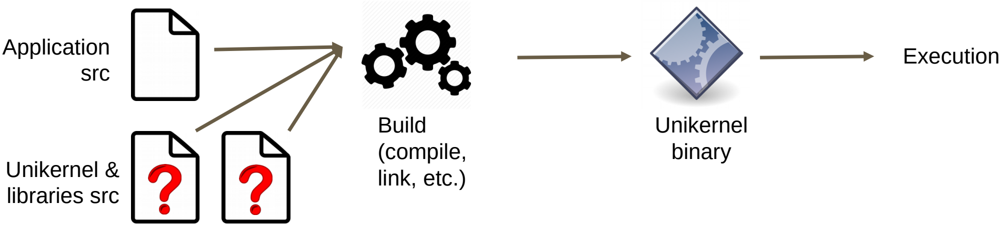
Why Porting to Unikernels is Hard



- Proprietary software → source code not available
- Incompatible language

HermiTux: a Unikernel Binary-Compatible with Linux

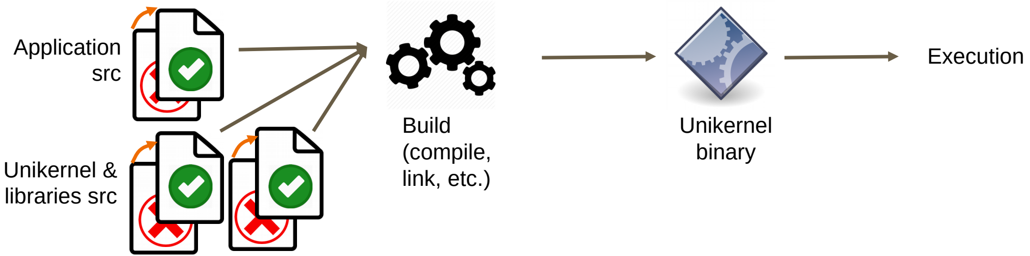
Why Porting to Unikernels is Hard



- Proprietary software → source code not available
- Incompatible language
- Unsupported features

HermiTux: a Unikernel Binary-Compatible with Linux

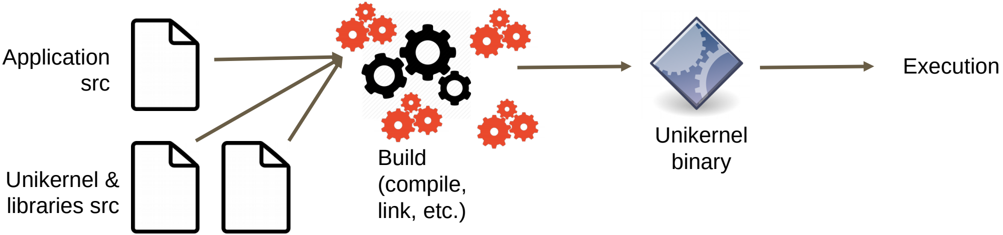
Why Porting to Unikernels is Hard



- Proprietary software → source code not available
- Incompatible language
- Unsupported features
- Porting is hard, needs knowledge about both application and unikernel

HermiTux: a Unikernel Binary-Compatible with Linux

Why Porting to Unikernels is Hard



- Proprietary software → source code not available
- Incompatible language
- Unsupported features
- Porting is hard, needs knowledge about both application and unikernel
- Complex build toolchains

HermiTux: a Unikernel Binary-Compatible with Linux

Why Porting to Unikernels is Hard

- Proprietary software → source code not available
- Incompatible language
- Unsupported features
- Porting is hard, needs knowledge about both application and unikernel
- Complex build toolchains

Our Solution: HermiTux

- **A unikernel *binary-compatible* with Linux applications**
  - I.e. a unikernel that can run application that have been compiled for Linux
  - For the x86-64 and aarch64 (ARM64) architectures

HermiTux: a Unikernel Binary-Compatible with Linux

Binary Compatibility with Linux Applications

HermiTux's objective: running as unikernels executables that have been compiled for a popular OS: Linux

Binary Compatibility with Linux Applications

HermiTux's objective: running as unikernels executables that have been compiled for a popular OS: Linux

- **Consider unmodified binaries** built with various compilers, from various source languages, potentially stripped/obfuscated
- **Don't assume access to applications' source code**
- **Consider both static and dynamic binaries**
- **Maintain unikernels principles and benefits**

HermiTux: a Unikernel Binary-Compatible with Linux

Binary Compatibility with Linux Applications

HermiTux's objective: running as unikernels executables that have been compiled for a popular OS: Linux

- **Consider unmodified binaries** built with various compilers, from various source languages, potentially stripped/obfuscated
- **Don't assume access to applications' source code**
- **Consider both static and dynamic binaries**
- **Maintain unikernels principles and benefits**

- Binary compatibility is achieved by developing a unikernel **conforming to the Linux Application Binary Interface (ABI)**
  - Convention, sets of rules a program needs to follow to execute on top of the Linux kernel
  - We need to implement the OS side of these rules in our unikernel

HermiTux: a Unikernel Binary-Compatible with Linux

The Linux ABI

It is partially architecture (ISA) specific, and is composed of two main sets of rules:

The Linux ABI

It is partially architecture (ISA) specific, and is composed of two main sets of rules:

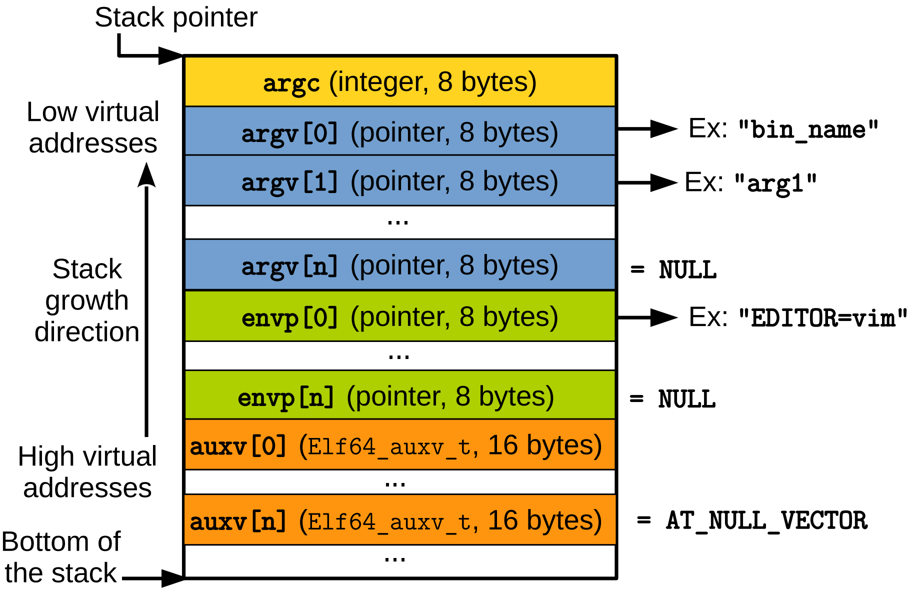
- **Load-time conventions**
  - Application binary format supported: ELF
  - What part of the address space is accessible for the application (currently lower half of the 48 bits virtual address space)
  - Layout of the application's stack & registers' content at the program entry point

The Linux ABI

It is partially architecture (ISA) specific, and is composed of two main sets of rules:

- **Load-time conventions**
  - Application binary format supported: ELF
  - What part of the address space is accessible for the application (currently lower half of the 48 bits virtual address space)
  - Layout of the application's stack & registers' content at the program entry point
- **Runtime conventions**
  - How to request services from the OS: system calls
  - Additional communication channels with the kernel through virtual filesystems (`/sys`, `/proc`, etc.) and shared memory areas (vDSO/vsyscall)

Load-Time Convention: Stack at Entry Point



Stack pointer

argc (integer, 8 bytes)

argv[0] (pointer, 8 bytes) → Ex: **"bin_name"**

argv[1] (pointer, 8 bytes) → Ex: **"arg1"**

...

argv[n] (pointer, 8 bytes) = **NULL**

envp[0] (pointer, 8 bytes) → Ex: **"EDITOR=vim"**

...

envp[n] (pointer, 8 bytes) = **NULL**

auxv[0] (`Elf64_auxv_t`, 16 bytes)

...

auxv[n] (`Elf64_auxv_t`, 16 bytes) = **AT_NULL_VECTOR**

...

Low virtual addresses

Stack growth direction

High virtual addresses

Bottom of the stack

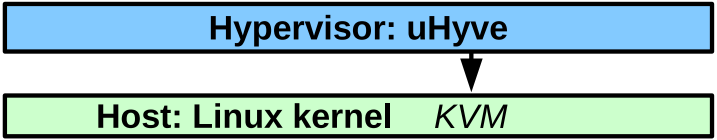
HermiTux: a Unikernel Binary-Compatible with Linux

Runtime Convention: System Calls

- Describes syscall invocation, parameters and return value passing.
- It is ISA-specific
- E.g. for x86-64:
    - Syscall identifier in `%rax`
    - Parameters are passed in orders in `%rdi`, `%rsi`, `%rdx`, `%r10`, `%r8`, `%r9`
    - Return value in `%rax`
    - Invocation with the `syscall` instruction

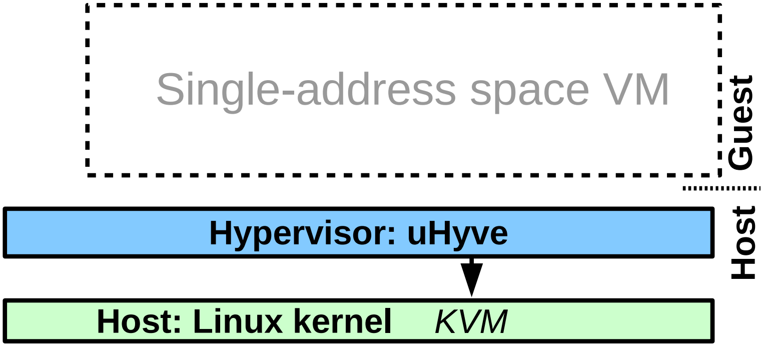
HermiTux: a Unikernel Binary-Compatible with Linux

Overview

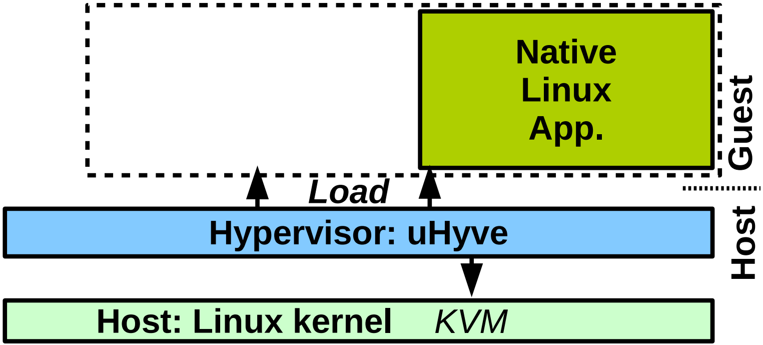


- Custom KVM-based hypervisor

Overview



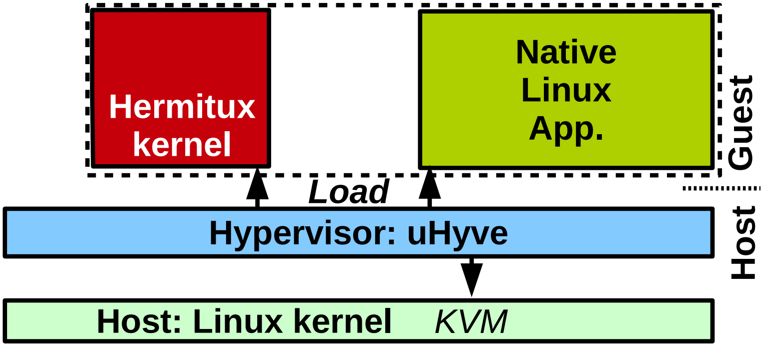
- Custom KVM-based hypervisor



- Custom KVM-based hypervisor

HermiTux: a Unikernel Binary-Compatible with Linux

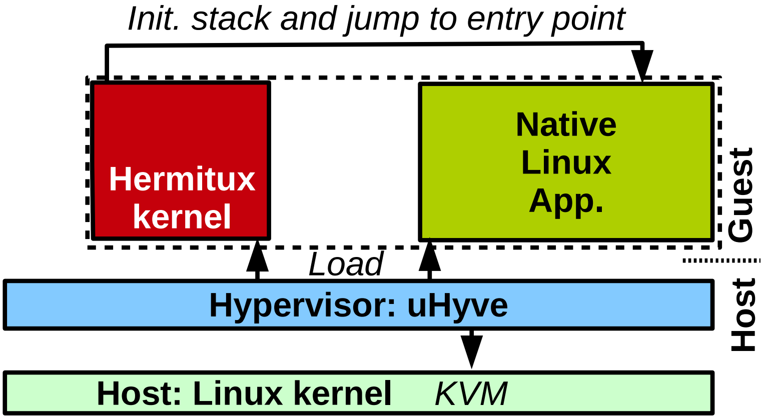
Overview



- Custom KVM-based hypervisor
- VMM loads app and kernel ELF binaries

HermiTux: a Unikernel Binary-Compatible with Linux

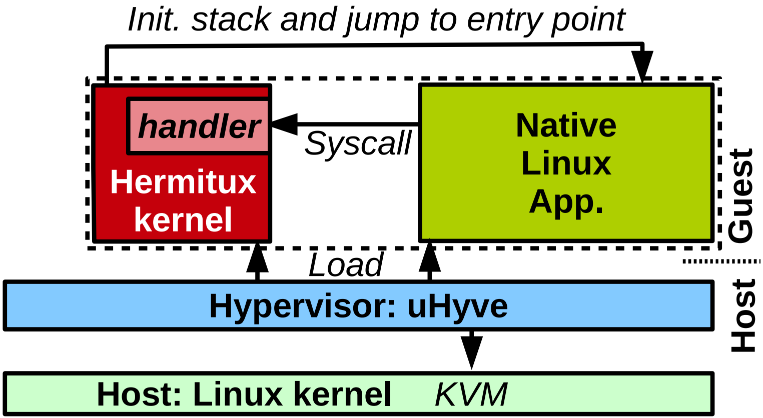
Overview



- Custom KVM-based hypervisor
- VMM loads app and kernel ELF binaries
- Follows load time ABI conventions

HermiTux: a Unikernel Binary-Compatible with Linux

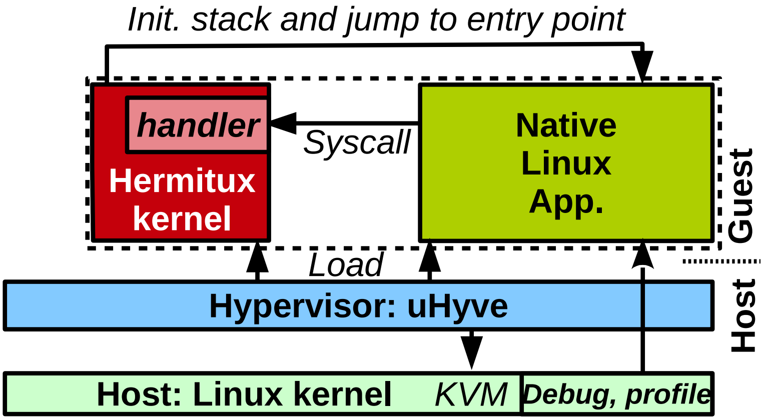
Overview



- Custom KVM-based hypervisor
- VMM loads app and kernel ELF binaries
- Follows load time ABI conventions
- And runtime conventions for syscalls

HermiTux: a Unikernel Binary-Compatible with Linux

Overview



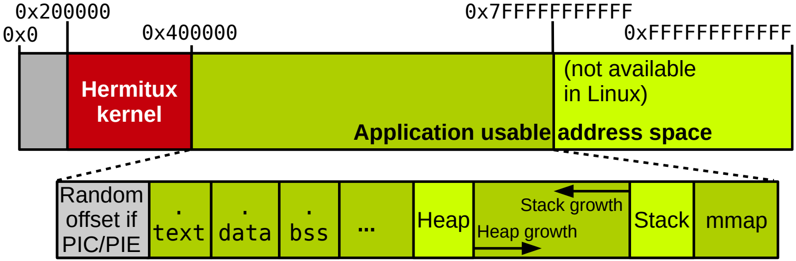
- Custom KVM-based hypervisor
- VMM loads app and kernel ELF binaries
- Follows load time ABI conventions
- And runtime conventions for syscalls

HermiTux: a Unikernel Binary-Compatible with Linux

Features: System Call Support

HermiTux supports 97 syscalls, 81 developed on top of the original HermitCore kernel. They offer support for:

- Statically and dynamically compiled binaries
- Multithreading (TLS, clone, synchronisation with futex)
- Signals
- Scheduling priorities
- Highly randomised memory mappings (entropy: 34 bits)



HermiTux: a Unikernel Binary-Compatible with Linux

Features: System Call Support (2)

Category	**System calls supported by HermiTux**
Filesystem	access, chdir, close, creat, faccessat, fcntl, fdatasync, fstat, fsync, getcwd, getdents, getdents64, lseek, lstat, mkdir, mkdirat, newfstatat, open, openat, pwrite64, read, readlinkat, readlink, readv, rmdir, stat, sync, syncfs, unlink, unlinkat, write, writev
Memory management	brk, sbrk, madvise, mincore, mmap, mprotect, mremap, munmap
Process management	clone*, exit, exit_group, getpid/ppid/gid/egid/euid/tid/uid, setsid, getrlimit, prlmit64, setrlimit, umask
Networking	accept, bind, connect, gethostname, getpeername, getsockname, getsockopt, listen, recvfrom, select, sendto, sendfile, sethostname, setsockopt, socket
Signals & Syncrhonization	futex, get_robust_list, kill, set_tid_address, rt_sigaction, sigaltstack, signal, tgkill, tkill
Scheduling	getprio, setprio, sched_getaffinity, sched_setaffinity, sched_yield
Time management	clock_getres, clock_gettime, gettimeofday, nanosleep, time
Miscellaneous	arch_prctl, ioctl, shutdown, sysinfo, uname

*threads only

Unikernel Benefits in HermiTux

The Issue

- Until now we described a small operating system that can run applications and bring these unikernels benefits
  - Security and isolation
  - Low memory/disk footprint
  - Fast boot time

What about the other unikernel benefits?

- Fast system calls (function calls)
- Kernel modularity

Unikernel Benefits in HermiTux

Fast Syscalls for Dynamic Binaries with Libc Substitution

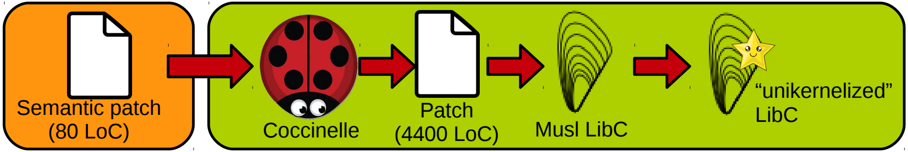
- HermiTux's syscall handler is invoked by the `syscall` instruction
    - Reintroduce high latency for system calls due to the exception

```
40127e:   b8 07 00 00 00          mov    $0x7,%eax (poll)
401283:   4c 89 c7                mov    %r8,%rdi
401286:   48 89 ca                mov    %rcx,%rdx
401289:   0f 05                   syscall
```

Unikernel Benefits in HermiTux

Fast Syscalls for Dynamic Binaries with Libc Substitution

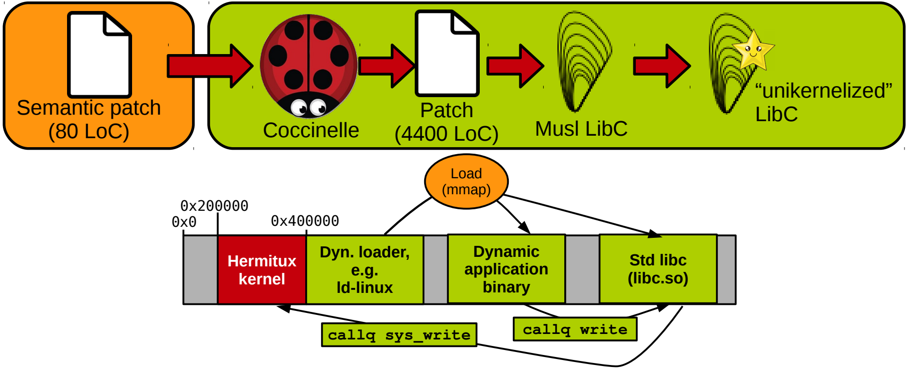
- For **dynamically compiled programs:**
  - At runtime **load a unikernel-aware Libc**
  - Custom Libc has system calls implemented as (fast) function calls directly into the kernel
  - Fork of Musl Libc, automatically transformed using Coccinelle



Unikernel Benefits in HermiTux

Fast Syscalls for Dynamic Binaries with Libc Substitution

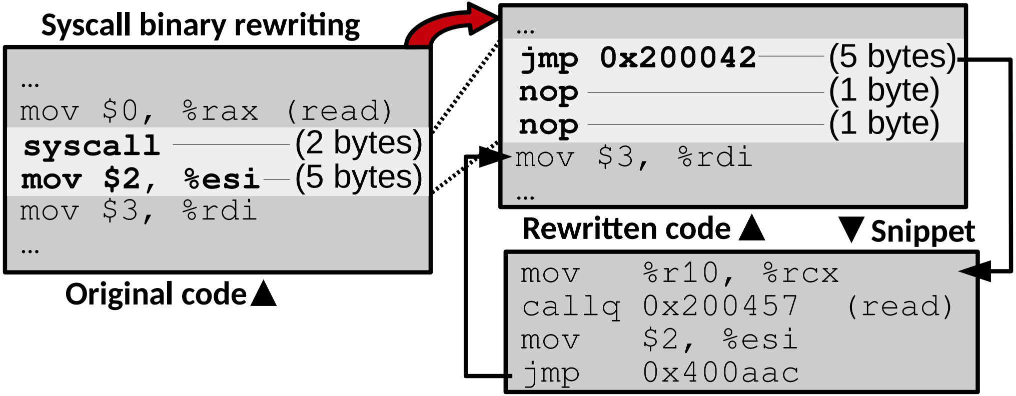
- For **dynamically compiled programs:**
  - At runtime **load a unikernel-aware Libc**
  - Custom Libc has system calls implemented as (fast) function calls directly into the kernel
  - Fork of Musl Libc, automatically transformed using Coccinelle



Unikernel Benefits in HermiTux

Fast Syscalls with Binary Rewriting

- What about **static binaries?**
- (Statically) **binary-rewrite `syscall` instructions to direct jumps to the syscall implementation**
  - Problem: `syscall` is 2 bytes long and any `call`/`jmp` instruction will be larger



Syscall binary rewriting

```
...
mov $0, %rax (read)
syscall             (2 bytes)
mov $2, %esi        (5 bytes)
mov $3, %rdi
...
```

Original code ▲

```
...
jmp 0x200042        (5 bytes)
nop                 (1 byte)
nop                 (1 byte)
mov $3, %rdi
...
```

Rewritten code ▲

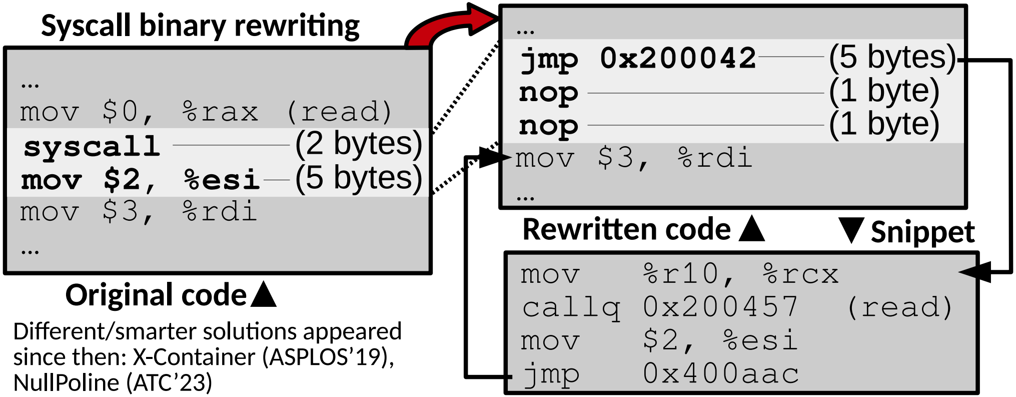
▼ Snippet

```
mov   %r10, %rcx
callq 0x200457  (read)
mov   $2, %esi
jmp   0x400aac
```

Unikernel Benefits in HermiTux

Fast Syscalls with Binary Rewriting

- What about **static binaries?**
- (Statically) **binary-rewrite `syscall` instructions to direct jumps to the syscall implementation**
    - Problem: `syscall` is 2 bytes long and any `call`/`jmp` instruction will be larger



Syscall binary rewriting

```
...
mov $0, %rax (read)
syscall          (2 bytes)
mov $2, %esi     (5 bytes)
mov $3, %rdi
...
```

Original code ▲

```
...
jmp 0x200042     (5 bytes)
nop              (1 byte)
nop              (1 byte)
mov $3, %rdi
...
```

Rewritten code ▲

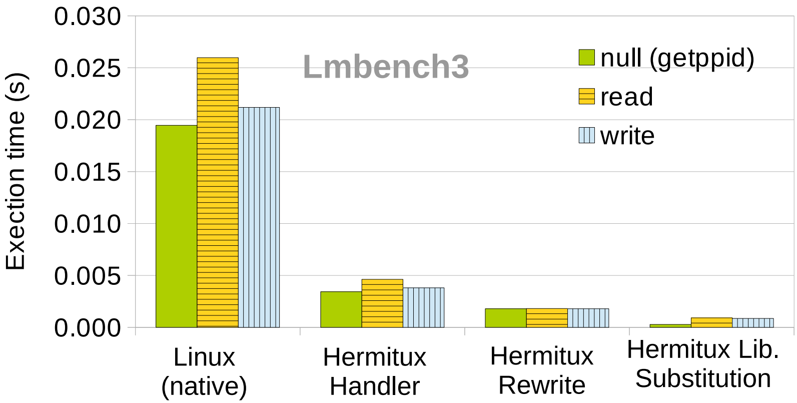
▼ Snippet

```
mov   %r10, %rcx
callq 0x200457   (read)
mov   $2, %esi
jmp   0x400aac
```

Different/smarter solutions appeared since then: X-Container (ASPLOS'19), NullPoline (ATC'23)

Unikernel Benefits in HermiTux

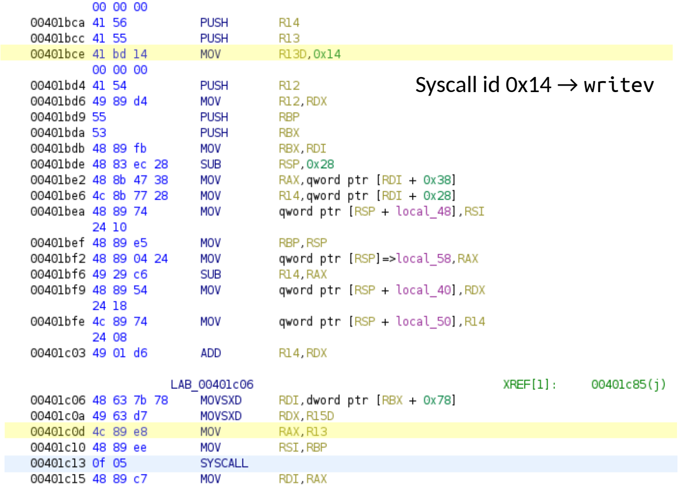
Fast Syscalls with Binary Rewriting



Unikernel Benefits in HermiTux

System-call-based Modularity

- System-call based modularity
    - Compile a kernel with support for only the necessary system calls
    - How to identify syscall needed without access to the sources?
    - **Use binary analysis** to find out what is the value in `%rax` (x86-64) or `%x8` (aarch64) for each `syscall` invocation



```
                 00 00 00
        00401bca 41 56           PUSH       R14
        00401bcc 41 55           PUSH       R13
        00401bce 41 bd 14        MOV        R13D,0x14
                 00 00 00
        00401bd4 41 54           PUSH       R12
        00401bd6 49 89 d4        MOV        R12,RDX
        00401bd9 55              PUSH       RBP
        00401bda 53              PUSH       RBX
        00401bdb 48 89 fb        MOV        RBX,RDI
        00401bde 48 83 ec 28     SUB        RSP,0x28
        00401be2 48 8b 47 38     MOV        RAX,qword ptr [RDI + 0x38]
        00401be6 4c 8b 77 28     MOV        R14,qword ptr [RDI + 0x28]
        00401bea 48 89 74        MOV        qword ptr [RSP + local_48],RSI
                 24 10
        00401bef 48 89 e5        MOV        RBP,RSP
        00401bf2 48 89 04 24     MOV        qword ptr [RSP]=>local_58,RAX
        00401bf6 49 29 c6        SUB        R14,RAX
        00401bf9 48 89 54        MOV        qword ptr [RSP + local_40],RDX
                 24 18
        00401bfe 4c 89 74        MOV        qword ptr [RSP + local_50],R14
                 24 08
        00401c03 49 01 d6        ADD        R14,RDX

                             LAB_00401c06                                  XREF[1]:     00401c85(j)
        00401c06 48 63 7b 78     MOVSXD     RDI,dword ptr [RBX + 0x78]
        00401c0a 49 63 d7        MOVSXD     RDX,R15D
        00401c0d 4c 89 e8        MOV        RAX,R13
        00401c10 48 89 ee        MOV        RSI,RBP
        00401c13 0f 05           SYSCALL
        00401c15 48 89 c7        MOV        RDI,RAX
```

Syscall id 0x14 → `writev`

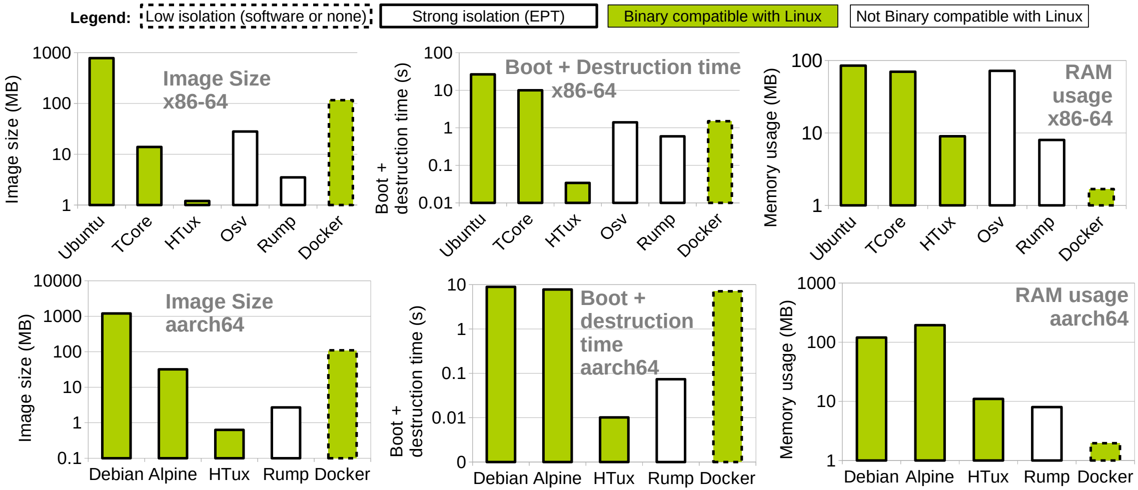
Unikernel Benefits in HermiTux

System-call-based Modularity

Program	Number of system calls	x86-64 kernel `.text` size reduction	aarch64 kernel `.text` size reduction
Minimal	4	21.22 %	29.26 %
Hello world	9	19.91 %	27.42 %
PARSEC Blackscholes	15	17.68 %	24.50 %
Postmark	27	16.02 %	22.55%
Sqlite	33	11.34 %	16.44%
Full syscalls support	97	-	-

HermiTux: a Unikernel Binary-Compatible with Linux

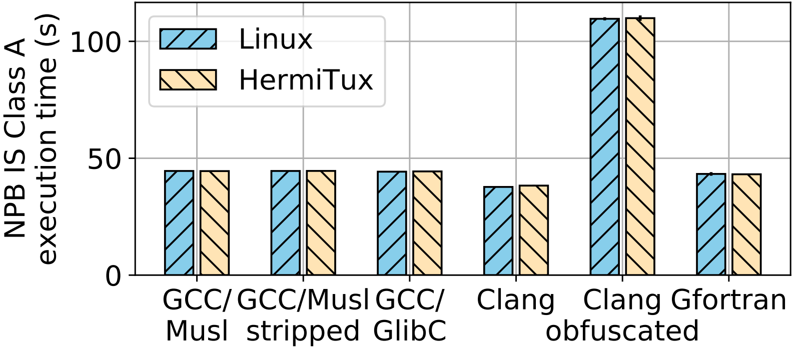
Evaluation



- Image 650x smaller, boot time 780x faster, RAM usage 9x lower than a Linux VM!

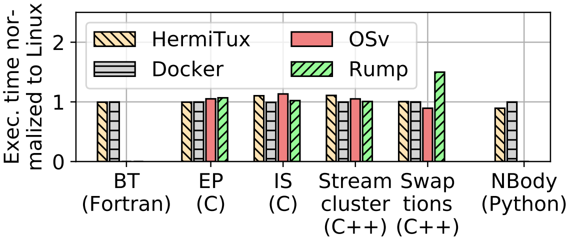
HermiTux: a Unikernel Binary-Compatible with Linux

Evaluation (2)



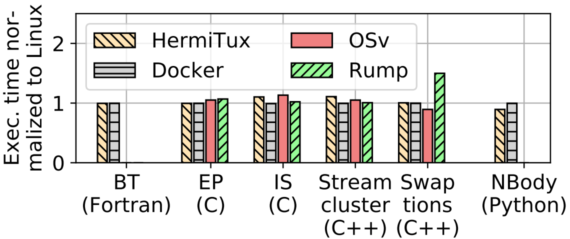
HermiTux: a Unikernel Binary-Compatible with Linux

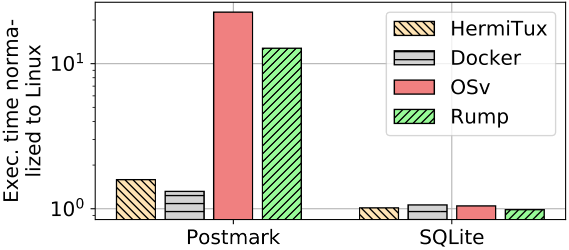
Evaluation (3)



HermiTux: a Unikernel Binary-Compatible with Linux

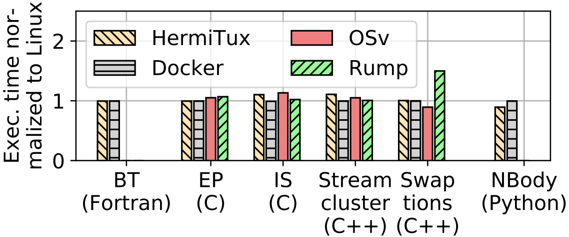
Evaluation (3)

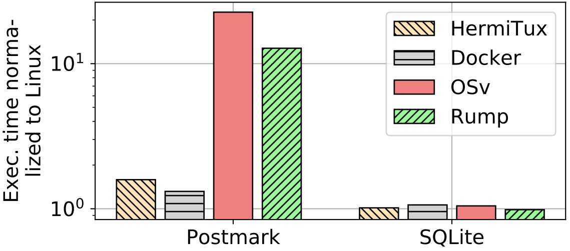


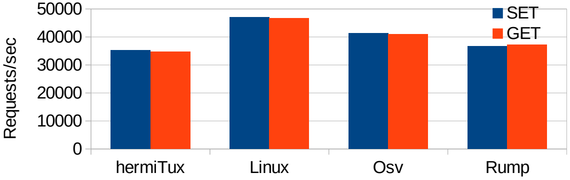


HermiTux: a Unikernel Binary-Compatible with Linux

Evaluation (3)

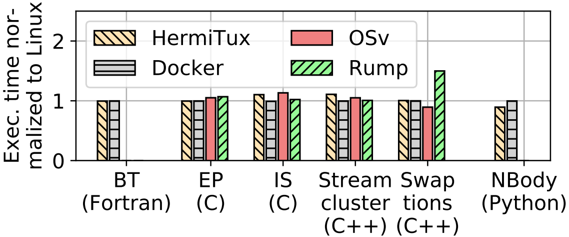


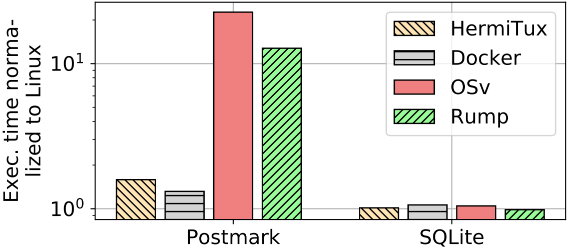


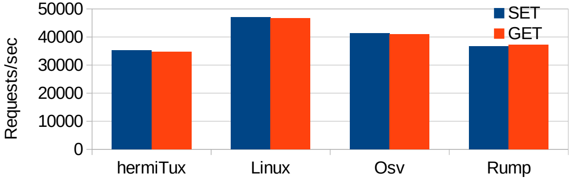


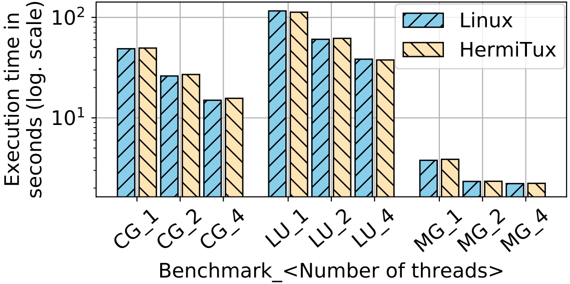
HermiTux: a Unikernel Binary-Compatible with Linux

Evaluation (3)









Demo

Table of contents

Introducing Unikernels

HermiTux: a Unikernel Binary-Compatible with Linux

FlexOS: an Operating System for Flexible Isolation

FlexOS: an Operating System for Flexible Isolation

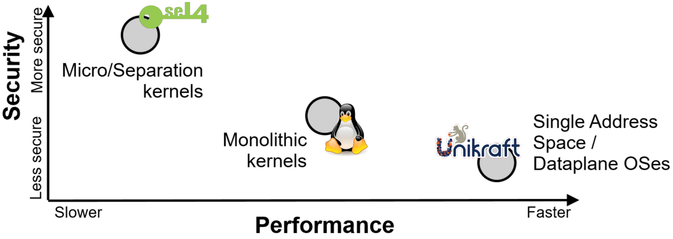
Motivation

- **OS security/isolation strategies are fixed at design time**
  - Isolation granularity, mechanisms used, data sharing strategies, etc.

FlexOS: an Operating System for Flexible Isolation

Motivation

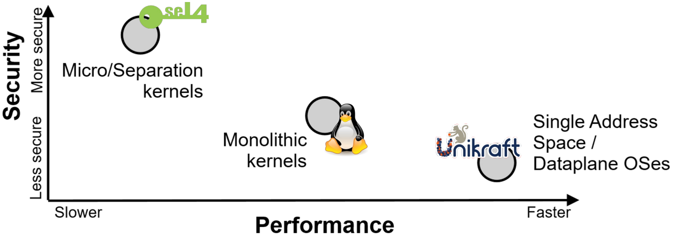
- **OS security/isolation strategies are fixed at design time**
  - Isolation granularity, mechanisms used, data sharing strategies, etc.



FlexOS: an Operating System for Flexible Isolation

Motivation

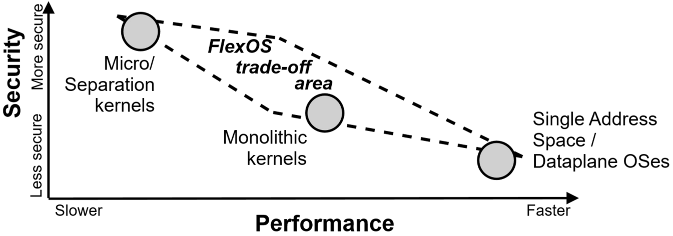
- **OS security/isolation strategies are fixed at design time**
  - Isolation granularity, mechanisms used, data sharing strategies, etc.
- Not really suitable in modern scenarios
  - Applications have heterogeneous needs in terms of security/performance
  - Application made of multiple components with various levels of trust
  - Machines support various isolation mechanisms, with new technologies underway
  - Hardware protection breaks (e.g. Meltdown)



FlexOS: an Operating System for Flexible Isolation

Motivation

- **OS security/isolation strategies are fixed at design time**
  - Isolation granularity, mechanisms used, data sharing strategies, etc.
- Not really suitable in modern scenarios
  - Applications have heterogeneous needs in terms of security/performance
  - Application made of multiple components with various levels of trust
  - Machines support various isolation mechanisms, with new technologies underway
  - Hardware protection breaks (e.g. Meltdown)



FlexOS

- LibOS that can specialise for **security**
- Security/isolation strategy can be instantiated at build time

FlexOS: an Operating System for Flexible Isolation

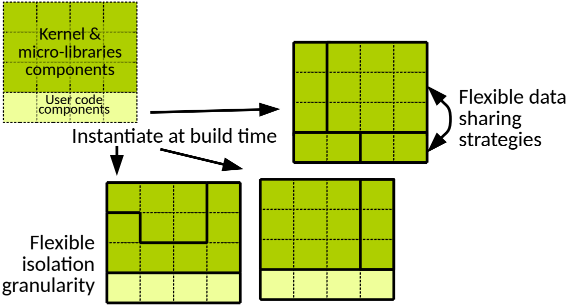
Flexible Builds

- Flexos decouple from the OS design important security/isolation decisions and allow selecting at build time:

FlexOS: an Operating System for Flexible Isolation

Flexible Builds

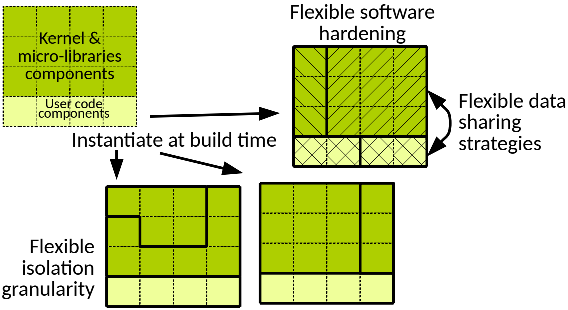
- Flexos decouple from the OS design important security/isolation decisions and allow selecting at build time:
  - **Compartmentalisation strategy and granularity**, as well as various **data sharing strategies** for communicating compartments


Kernel & micro-libraries components
User code components
Instantiate at build time
Flexible data sharing strategies
Flexible isolation granularity

FlexOS: an Operating System for Flexible Isolation

Flexible Builds

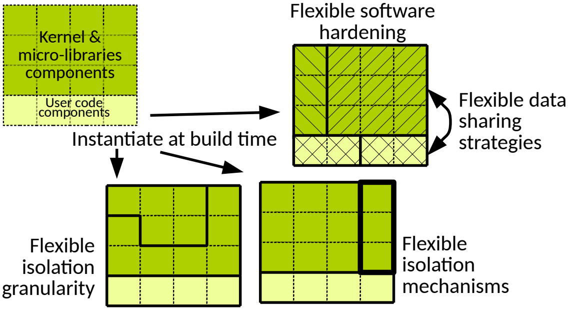
- Flexos decouple from the OS design important security/isolation decisions and allow selecting at build time:
  - **Compartmentalisation strategy and granularity**, as well as various **data sharing strategies** for communicating compartments
  - Different levels of per-compartment **software hardening** (ASan, etc.)



FlexOS: an Operating System for Flexible Isolation

Flexible Builds

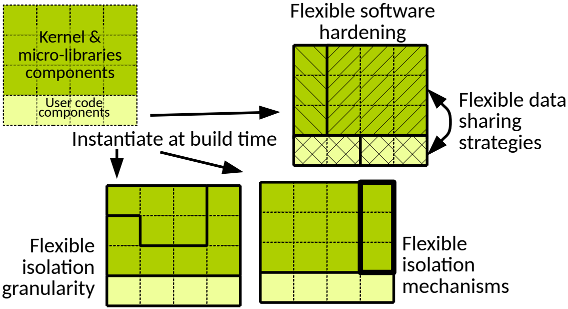
- Flexos decouple from the OS design important security/isolation decisions and allow selecting at build time:
  - **Compartmentalisation strategy and granularity**, as well as various **data sharing strategies** for communicating compartments
  - Different levels of per-compartment **software hardening** (ASan, etc.)
  - Various **isolation mechanisms** to enforce the compartmentalisation: MPK, EPT


Kernel & micro-libraries components
User code components
Instantiate at build time
Flexible software hardening
Flexible data sharing strategies
Flexible isolation granularity
Flexible isolation mechanisms

FlexOS: an Operating System for Flexible Isolation

Flexible Builds

- Flexos decouple from the OS design important security/isolation decisions and allow selecting at build time:
  - **Compartmentalisation strategy and granularity**, as well as various **data sharing strategies** for communicating compartments
  - Different levels of per-compartment **software hardening** (ASan, etc.)
  - Various **isolation mechanisms** to enforce the compartmentalisation: MPK, EPT


Kernel & micro-libraries components
User code components
Instantiate at build time
Flexible software hardening
Flexible data sharing strategies
Flexible isolation granularity
Flexible isolation mechanisms

How to enable flexible builds?

FlexOS: an Operating System for Flexible Isolation

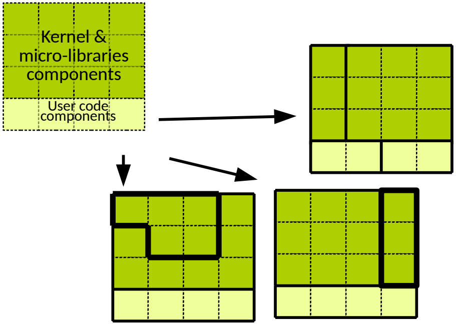
Enabling Flexible Builds

- LibOS **compartmentalised to the finest level**: components
- Compartmentalisation API: annotations in the code marking **components interfaces** and **shared data**

FlexOS: an Operating System for Flexible Isolation

Enabling Flexible Builds

- LibOS **compartmentalised to the finest level**: components
- Compartmentalisation API: annotations in the code marking **components interfaces** and **shared data**
- **Code transformations** at build time merging/separating components into compartments and instantiating gates



Enabling Flexible Builds: Porting API

```
    int rc, connfd;
    char buf[512];
    /* … */
    rc = recv(connfd, buf, 512, 0);
```

Enabling Flexible Builds: Porting API

```
int rc, connfd;
char buf[512];
/* … */
rc = recv(connfd, buf, 512, 0);
```

Porting

```
int rc, connfd;
char buf[512] __attribute__((flexos_share));
/* … */
rc = flexos_gate(liblwip, recv, connfd, buf, 512, 0);
```

FlexOS: an Operating System for Flexible Isolation

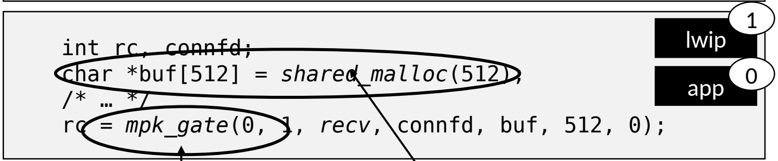
Enabling Flexible Builds: Porting API

```
int rc, connfd;
char buf[512];
/* … */
rc = recv(connfd, buf, 512, 0);
```

Porting

```
int rc, connfd;
char buf[512] __attribute__((flexos_share));
/* … */
rc = flexos_gate(liblwip, recv, connfd, buf, 512, 0);
```

Automatic gate instantiation at build time



```
int rc, connfd;
char *buf[512] = shared_malloc(512);
/* … */
rc = mpk_gate(0, 1, recv, connfd, buf, 512, 0);
```

Replace with shared heap allocation

Replace with MPK gate

Coccinelle

Enabling Flexible Builds: Porting API

```
int rc, connfd;
char buf[512];
/* … */
rc = recv(connfd, buf, 512, 0);
```

Porting

```
int rc, connfd;
char buf[512] __attribute__((flexos_share));
/* … */
rc = flexos_gate(liblwip, recv, connfd, buf, 512, 0);
```

Automatic gate instantiation at build time

lwip + app

```
int rc, connfd;
char buf[512];
/* … */
rc = recv(connfd, buf, 512, 0);
```

Replace with standard stack allocation and function call

Coccinelle

FlexOS: an Operating System for Flexible Isolation

A Few Numbers

Redis throughput on a total of 80 configuration, isolation with MPK, varying

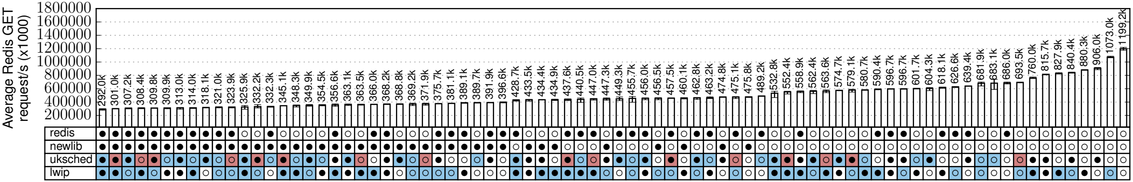
- Number of compartments (1, 2, 3)
- The distribution of software components in compartments
- Software hardening (per compartment, on/off)

FlexOS: an Operating System for Flexible Isolation

A Few Numbers

Redis throughput on a total of 80 configuration, isolation with MPK, varying

- Number of compartments (1, 2, 3)
- The distribution of software components in compartments
- Software hardening (per compartment, on/off)



FlexOS: an Operating System for Flexible Isolation

Design Space Exploration

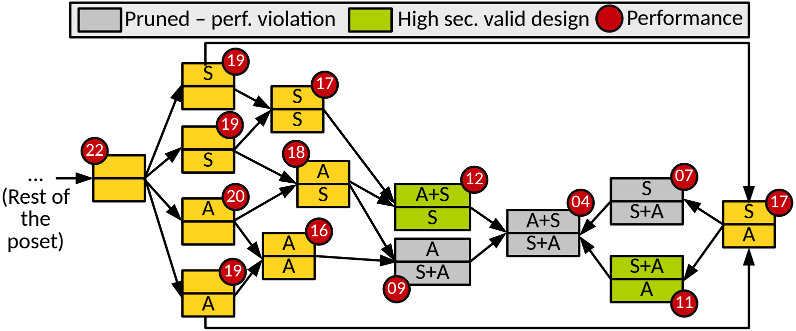
- How to explore the wide design space FlexOS gives access to?
- Security is hard to quantify
- Propose a **semi-automated exploration strategy** based on partially ordered sets

FlexOS: an Operating System for Flexible Isolation

Design Space Exploration

- How to explore the wide design space FlexOS gives access to?
- Security is hard to quantify
- Propose a **semi-automated exploration strategy** based on partially ordered sets

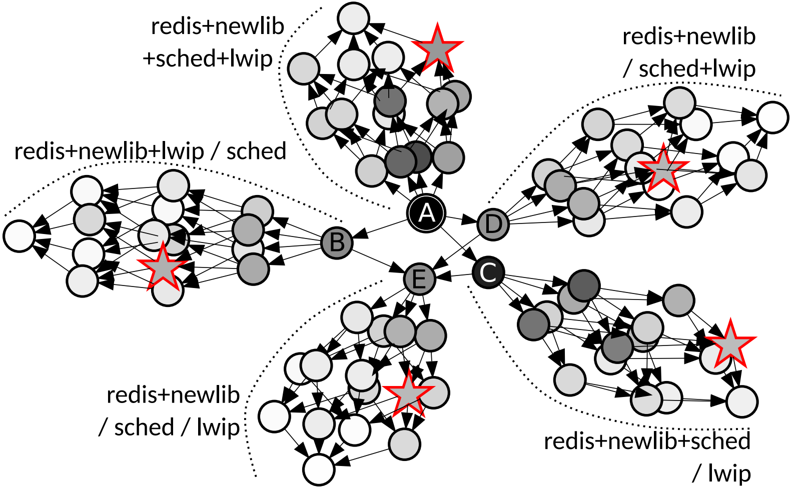
Example with 2 compartments and 2 software hardening techniques:
safe stack (S) and ASan (A):



FlexOS: an Operating System for Flexible Isolation

Design Space Exploration

Poset design space exploration for Redis:



Conclusion

Availability and Publications

Everything is open source!

- **HermiTux**
  - `https://ssrg-vt.github.io/hermitux/`
  - P. Olivier, D. Chiba, S. Lankes, C. Min and B. Ravindran, **A Binary-Compatible Unikernel**, *VEE'19*
  - P. Olivier, H. Lefeuvre, D. Chiba, S. Lankes, C. Min and B. Ravindran, **A Syscall-Level Binary-Compatible Unikernel**, *IEEE TC*, 2021
- **FlexOS**
  - `https://project-flexos.github.io/`
  - H. Lefeuvre, V. Badoiu, A. Jung, S Teodorescu, S. Rauch, F. Huici, C. Raiciu, and P. Olivier, **FlexOS: Towards Flexible OS Isolation**, *ASPLOS'22*
  - H. Lefeuvre, V. Badoiu, S Teodorescu, P. Olivier, T. Mosnoi, R. Deaconescu, F. Huici, and C. Raiciu, **FlexOS: Making OS Isolation Flexible**, *HotOS'21*
  - **FlexOS is part of Hugo Lefeuvre's PhD work**: `https://www.owl.eu.com/research.html`

Please do not hesitate to get in touch :) pierre.olivier@manchester.ac.uk