再利用可能なモノリシックカーネル

Hajime Tazaki

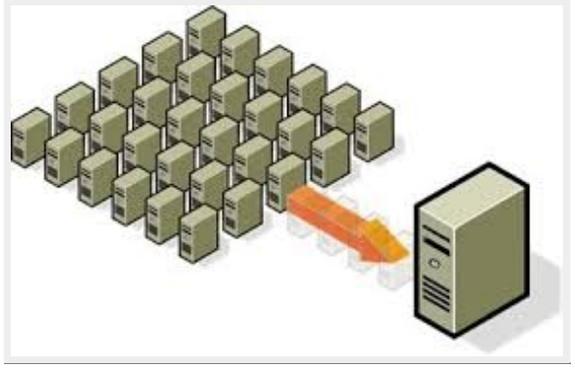
IIJ Innovation Institute

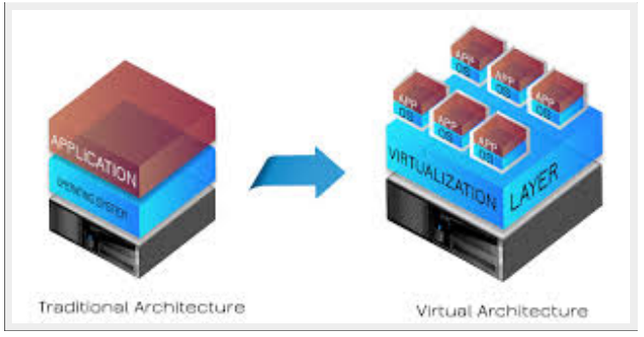
2016/04/12, IIJlab セミナー

背景

- 仮想化技術の発達
- リソース (計算機・ストレージ・ネットワークなど)の抽象化
- 物理的制約の緩和、複雑性の減少など

仮想化する事で、できなかった事をできるようにする





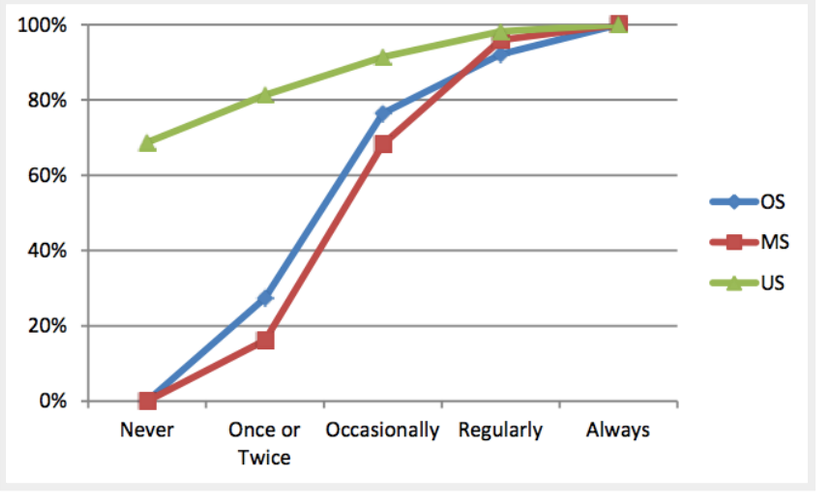
- http://ableit.ca/business-support-services/server-virtualization/
- http://exelos.com/solutions/virtualization/

様々な仮想化とその用途 (ネットワーク)

- 手法
  - 疑似 (仮想マシン)
  - 名前空間分離
  - カーネル迂回
- 用途
  - 多重化、隔離環境
  - 機能追加への自由度
  - 性能改善
  - 軽量化、分化・専用用途
  - デバッグ・テスト、詳細・再現調査
  - 大規模ネットワーク実験

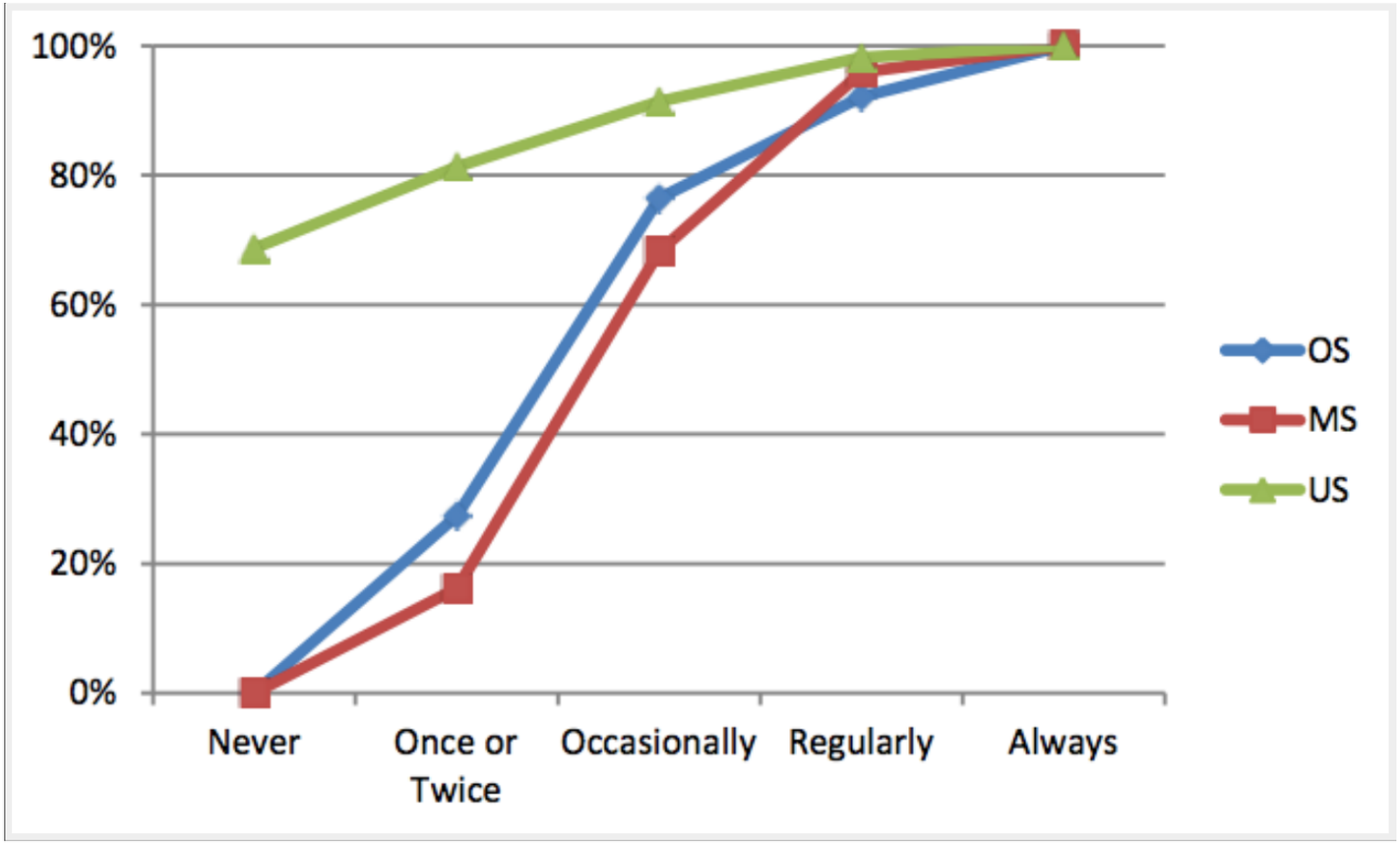
モチベーション (Why)

- 新しい用途でも、実績のあるソフトウェアが利用したい
  - ネットワークスタック (618K LoC, Linux)
  - ファイルシステム (752K LoC, Linux)
  - アプリケーション (a lot)
- VM は重い (再現性向上 => 機動性低下)
  - 抽象化の場所 (プロセッサ、OS、システムコール、etc)
- 仮想化技術から前進させて、変形させる



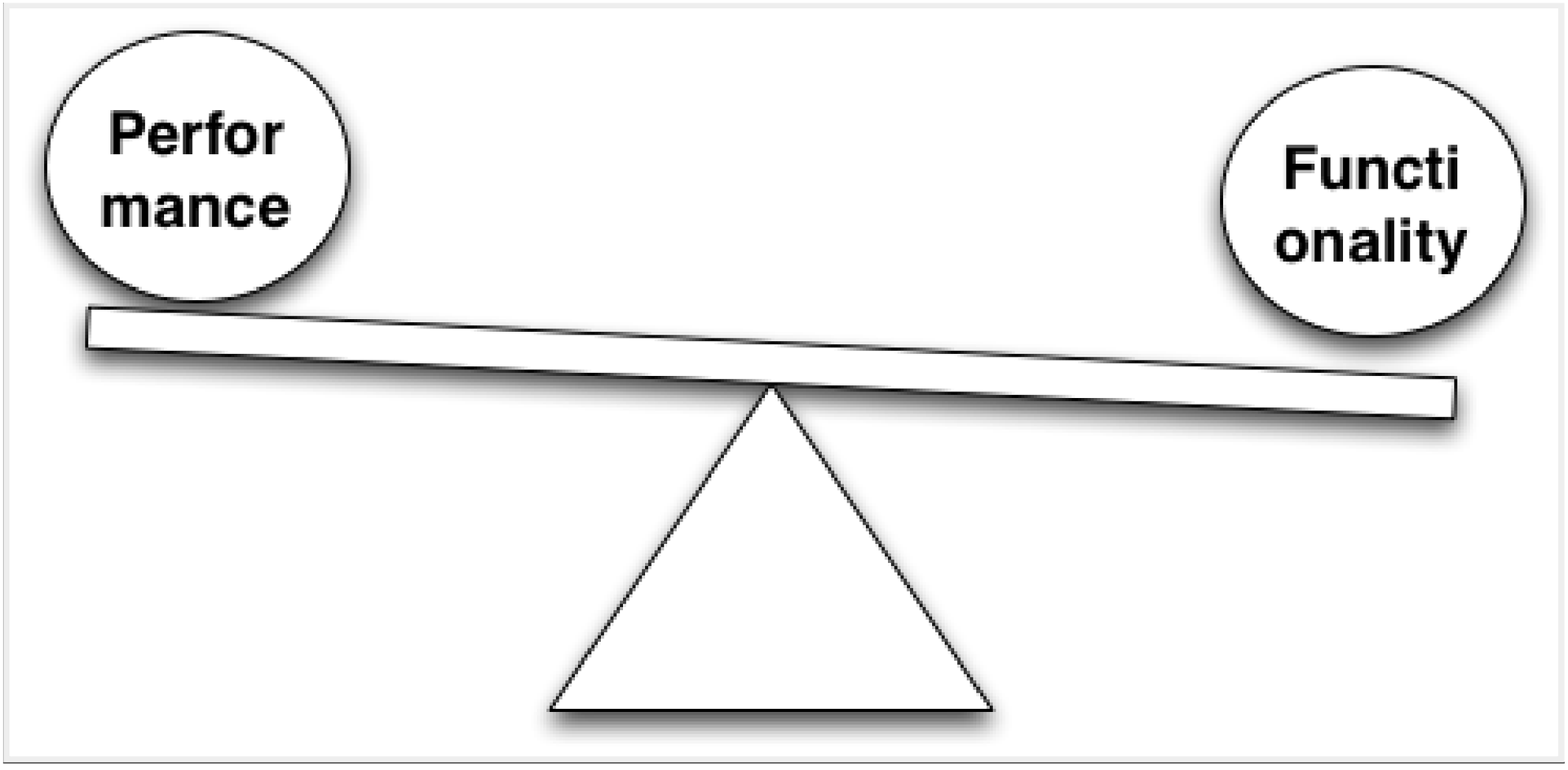
Poll: "When you download and run software, how often do you use a virtual machine (to reduce security risks)?"

- Jon Howell, Galen Hunt, David Molnar, and Donald E. Porter, Living Dangerously: A Survey of Software Download Practices, no. MSR-TR-2010-51, May 2010



Poll: "When you download and run software, how often do you use a virtual machine (to reduce security risks)?"

- Jon Howell, Galen Hunt, David Molnar, and Donald E. Porter, Living Dangerously: A Survey of Software Download Practices, no. MSR-TR-2010-51, May 2010



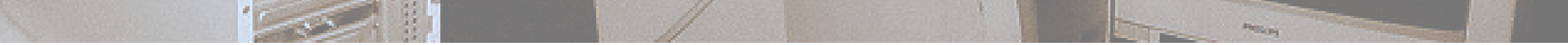
性能と利便性の両立 ？

- 性能を求めるには既存の資産(コード)は障壁になる
- 既存の資産を生かすと、性能を犠牲にしなければならない

*Any problem in computer science can be solved with another level of **indirection**.*

(Wheeler and/or Lampson)

img src: https://www.flickr.com/photos/thomasclaveirole/305073153



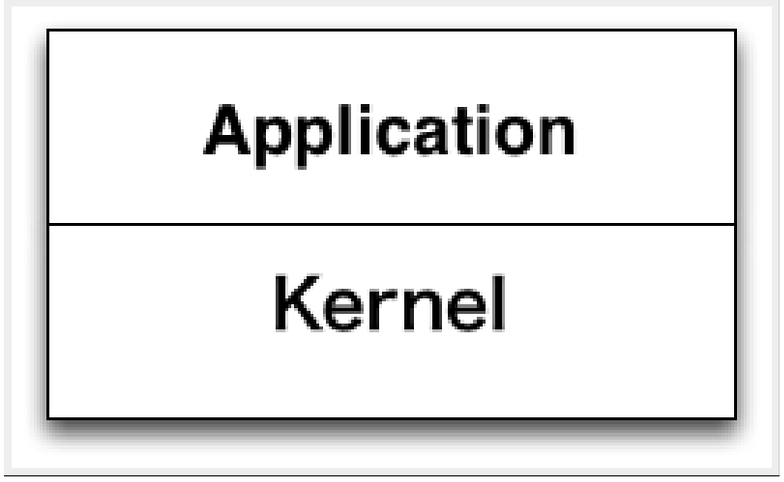
モノリシックカーネルの再利用とは (What ?)

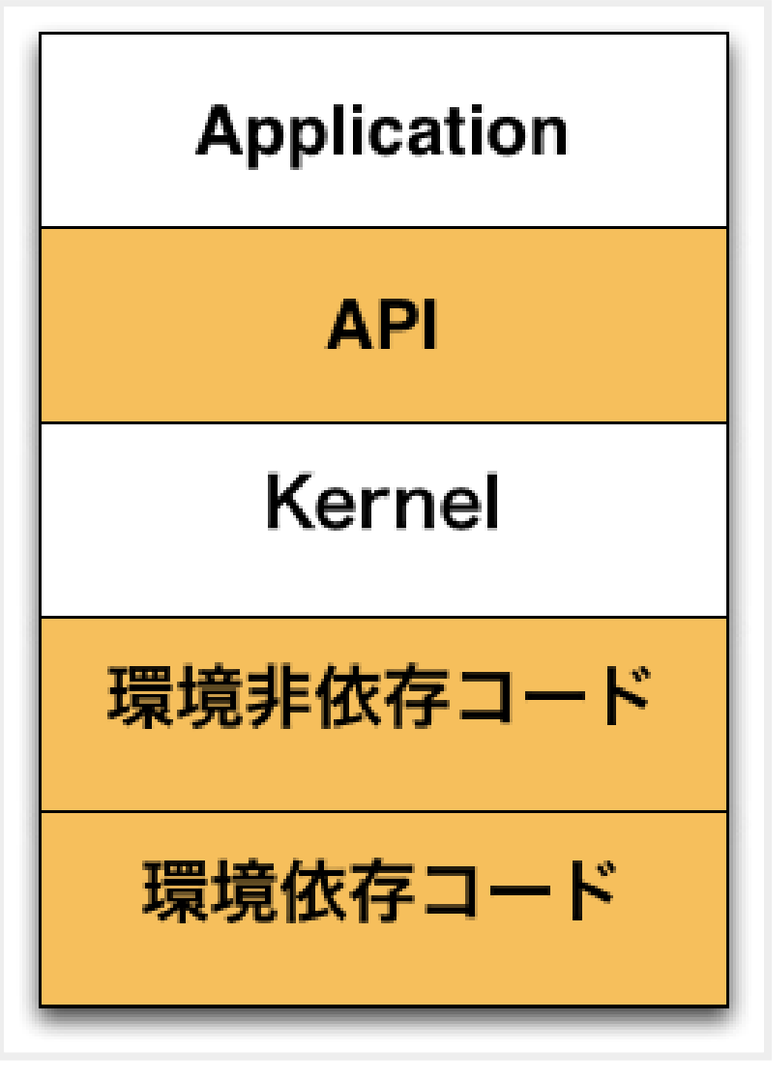
- Anykernel: NetBSD rump kernel に由来

> *We define an anykernel to be an organization of kernel code which allows the kernel's **unmodified** drivers to be **run in various configurations** such as application libraries and microkernel style servers, and also as part of a monolithic kernel. -- Kantee 2012.*

- 実績あるモノリシックカーネルのソースコードをそのまま利用 (unmodified)
- 追加具材を糊付けし変形させ、ライブラリ化

モノリシックカーネルの再利用とは (cont'd)





- 機能のアウトソース (環境依存コード)
  - 時計
  - メモリ
  - スケジュール
  - デバイス (NIC/ブロック) I/O
- 様々な実装でこの仕組みを利用
  - SeaStar/OSv, mTCP, LKL/LibOS, rumpkernel
- Anykernel として作るには、カーネル内に*透過的*にサブシステムを導入する必要があり
  - 環境**非依存**コード
  - CPU architecture として
  - 独立したサブシステムとして

Who else ?

- Full scratch (ゼロから実装)
  - Mirage [ASPLOS 2013]
  - IncludeOS [CloudCom 2016]
  - SeaStar
  - mTCP [NSDI 2014]
- porting (既存 OS の移植)
  - OSv [USENIX 2014]
  - libuinet
  - SandStorm [SIGCOMM 2014]
- Anykernel
  - NetBSD rump [USENIX 2009]
  - UML
  - DrawBridge (?)

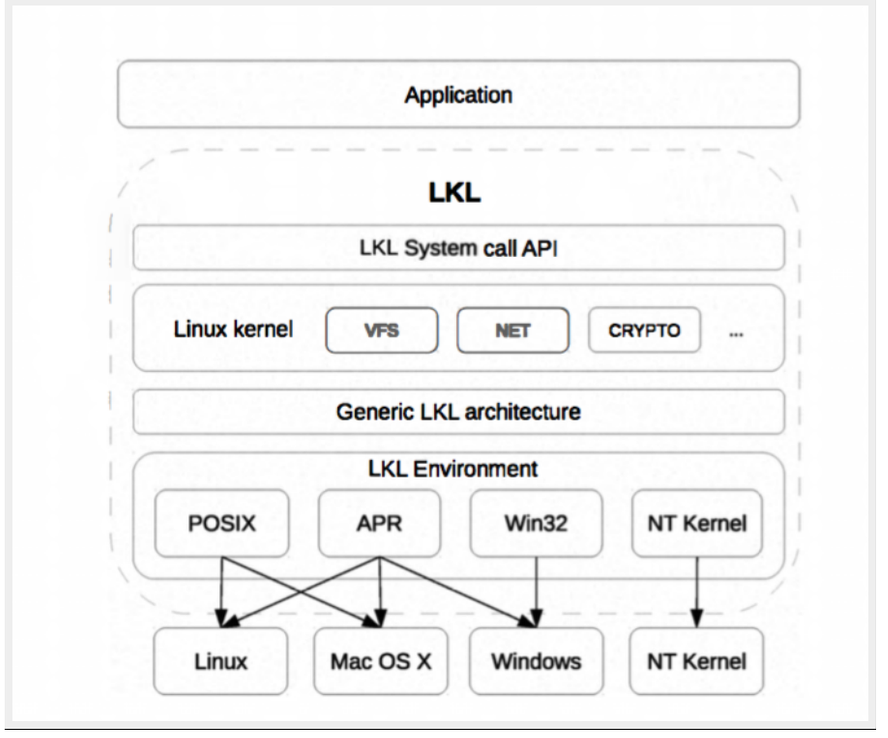
特定用途カーネルの作り方

	性能	既存資産	最新版追従
full scratch	++	--	+
porting	+	+/-	--
anykernel	-	++	++

Linux Kernel Library (LKL)

Linux Kernel Library

- Linux カーネルのライブラリ
- ハードウェア非依存のアーキテクチャ
- 下位ホストのインターフェースを規定
  - 依存部をアウトソース
  - Windows, Linux, FreeBSDで動作
- デバイスI/Oを簡素化
  - virtio ホスト実装
  - Linux カーネル内のドライバ利用

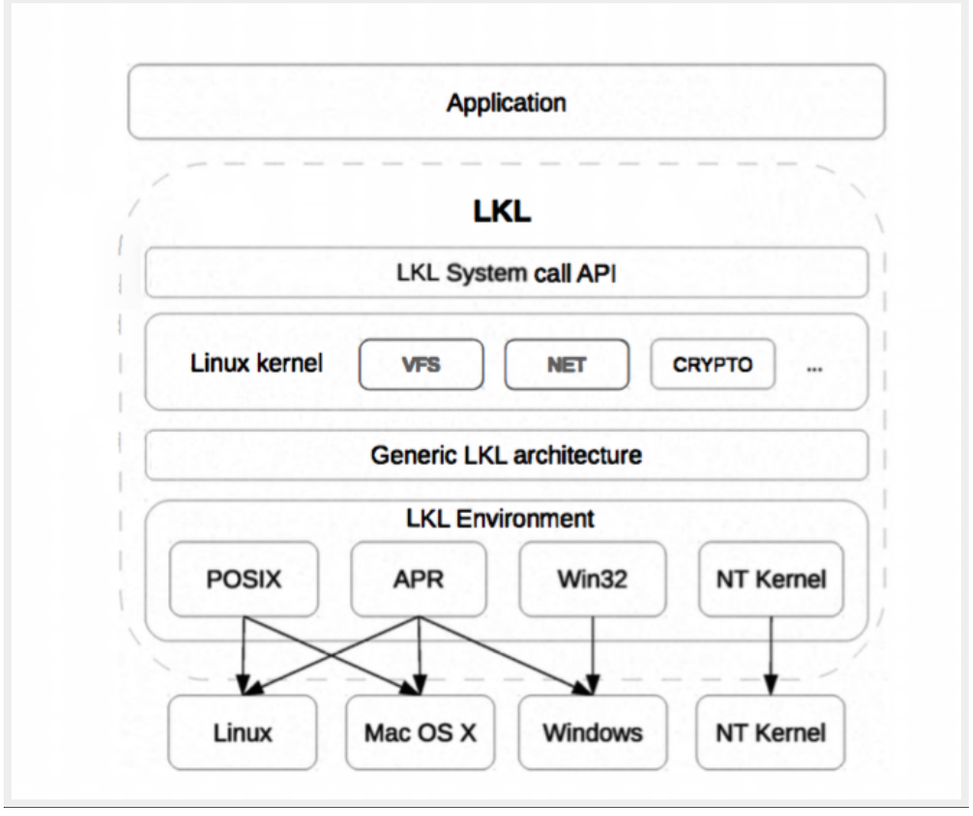


Purdila et al., LKL: The Linux kernel library, RoEduNet 2010.

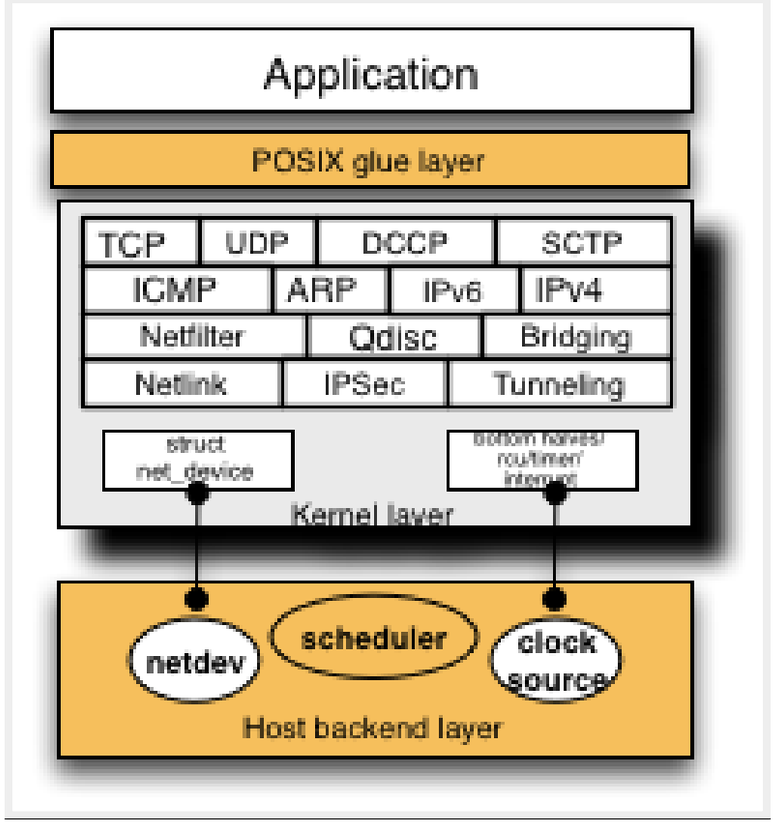
歴史

- rump: 2007 (NetBSD)
- LKL: 2007 (Linux)
- DCE/LibOS: 2008 (Linux/FreeBSD)
- LibOS/LKL revival: 2015
  - LibOS merged to LKL

LKL v.s. LibOS



LKL

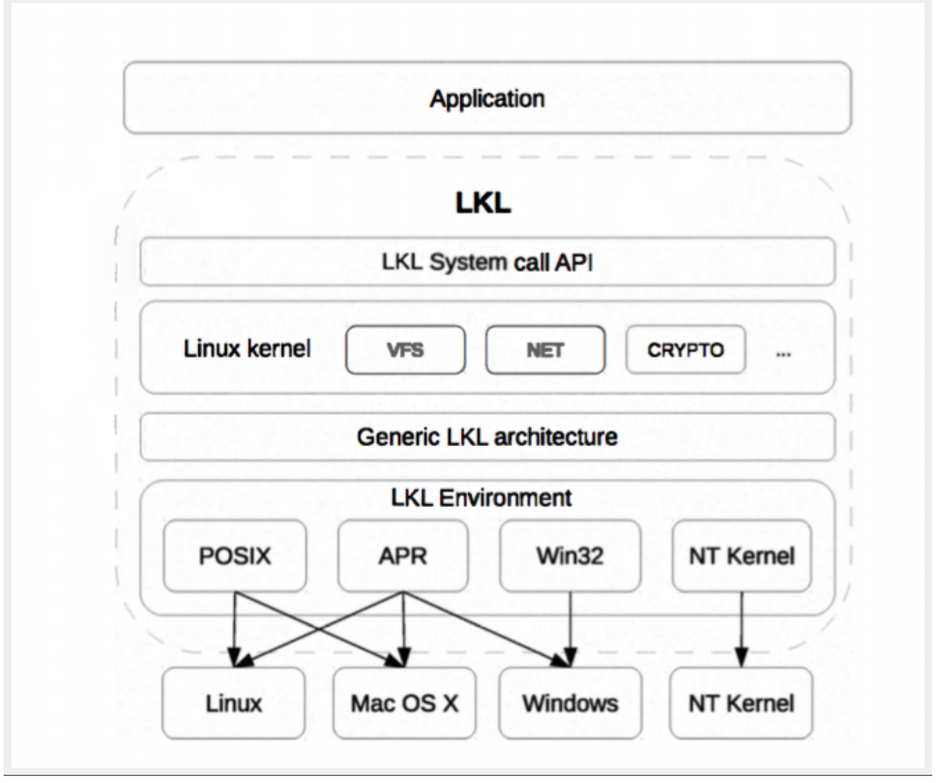


LibOS

LKL v.s. LibOS (cont'd)

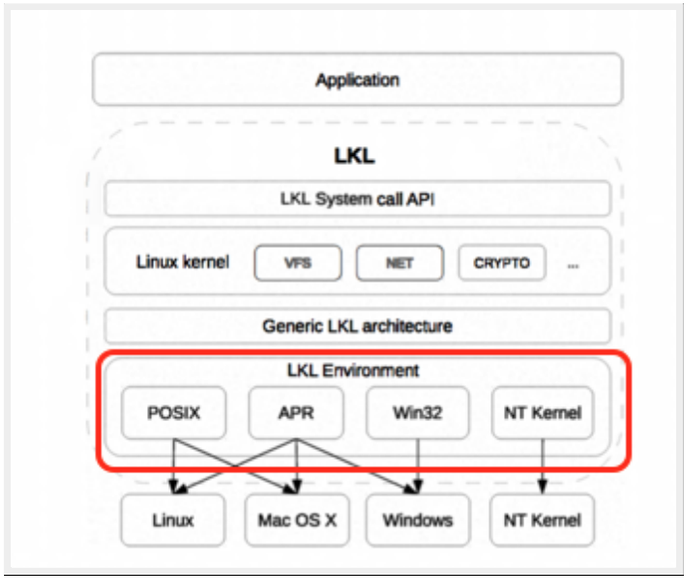
- LoC:
  - arch/lkl (LKL) < arch/lib (LibOS)
  - スタブコードの量に依存
- 共通点
  - カーネルコンテキストの表現 (POSIX thread)
  - ホストインターフェース (時計、メモリ、スケジュール)
  - CPU 非依存の arch として実装
  - 他カーネルコードへの修正なし
- 相違点
  - LibOS: 高位 API (timer, irq, kthread)を pthread で再実装
  - LKL: pthread で IRQ, kthread, timer などを API 再実装はせずに表現

内部構造



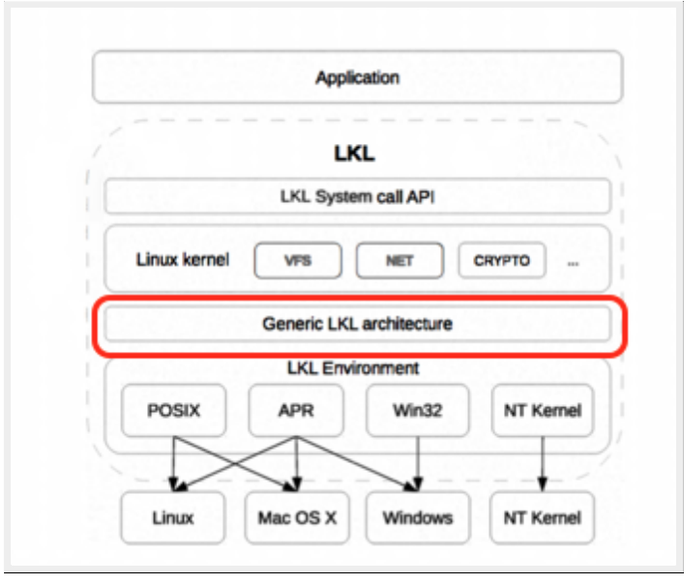
1. ホストバックエンド (host_ops)
2. CPU 非依存アーキテクチャ (arch/lkl)
3. アプリケーションインターフェース

1. ホストバックエンド



- 環境依存部のインターフェース
  - 異プラットフォーム間で共通化
  - (rump ハイパーコール like)
- Virtio によるデバイスインターフェース
  - ブロックデバイス <=> ディスクイメージ
  - ネットワーク <=> TAP, DPDK, VDE

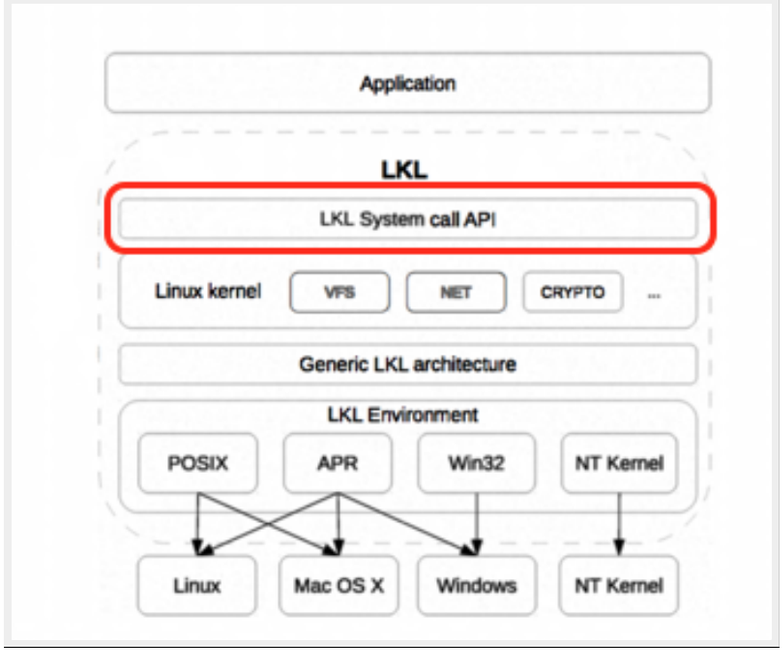
2. CPU 非依存アーキテクチャ



アーキテクチャ (arch/lkl)

- CPU アーキテクチャと同等に扱える
  - 他コードへの影響をなくす
- 2400 LoC
  - スレッド構造体 (struct thread_info) 定義
  - irq, timer, syscall handler を実装
  - 下位層リソースへのアクセス(環境依存)は、host_opsにて表現

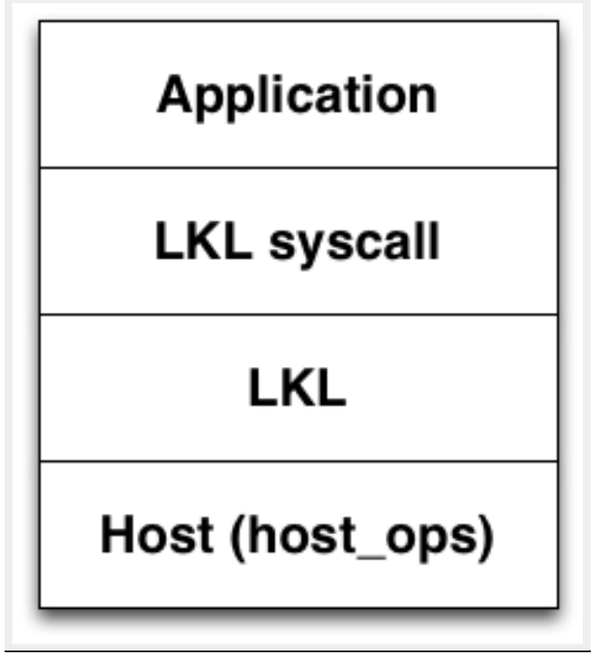
3. アプリケーションインターフェース



- Case 1: use exposed API (LKL syscall)
- Case 2: use host libc (LD_PRELOAD)
- Case 3: extend (alternative) libc

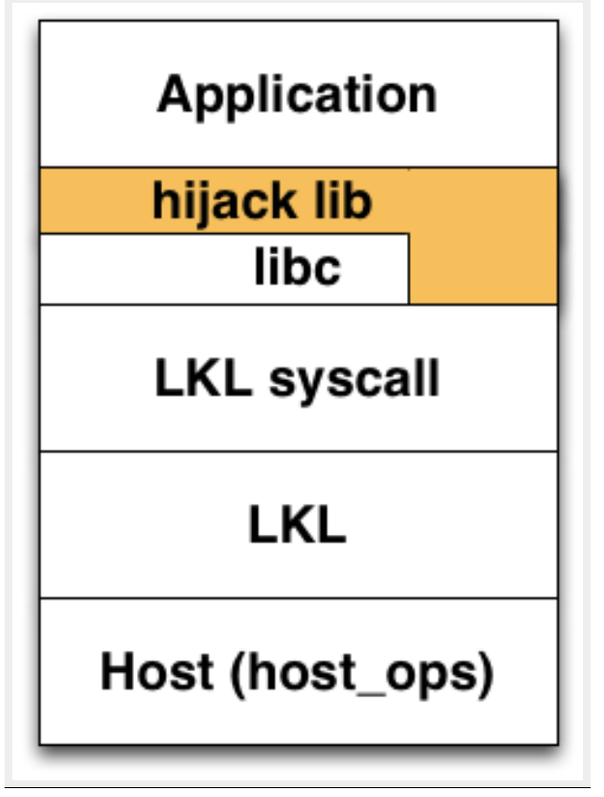
Case 1: use exposed API (LKL syscall)

- カーネル内のシステムコール入り口を直接 call
  - *lkl_sys_open()*, *lkl_sys_socket()*
- 通常のシステムコールとほぼ同じ
  - 復帰値、エラー番号の通知が違う
- アプリケーションはこの API と、ホストの持つ API を両方利用可能
  - LKL で ext4 ファイルを lkl_sys_read() => ホスト (Windows) で write()



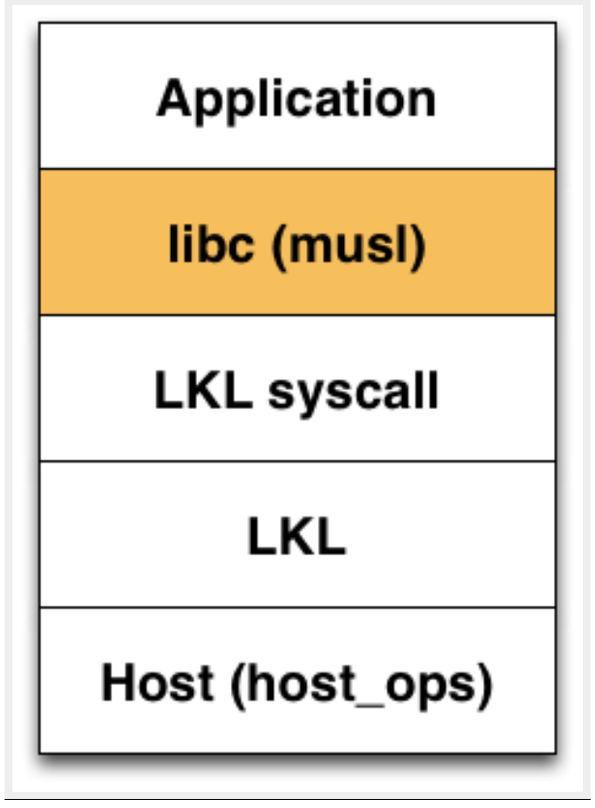
Case 2: ホスト標準ライブラリのハイジャック

- ホストの標準ライブラリ (libc) を
  LKL システムコールに実行時に置きかえ
  - LD_PRELOAD
  - socket() => lkl_sys_socket()
- ホストのプログラムバイナリをそのまま利用可能
- 置きかえ可能なシンボルに制限あり
- 非 Linux ホストでのシステムコール変換は必要



Case 3: extend (alternative) libc

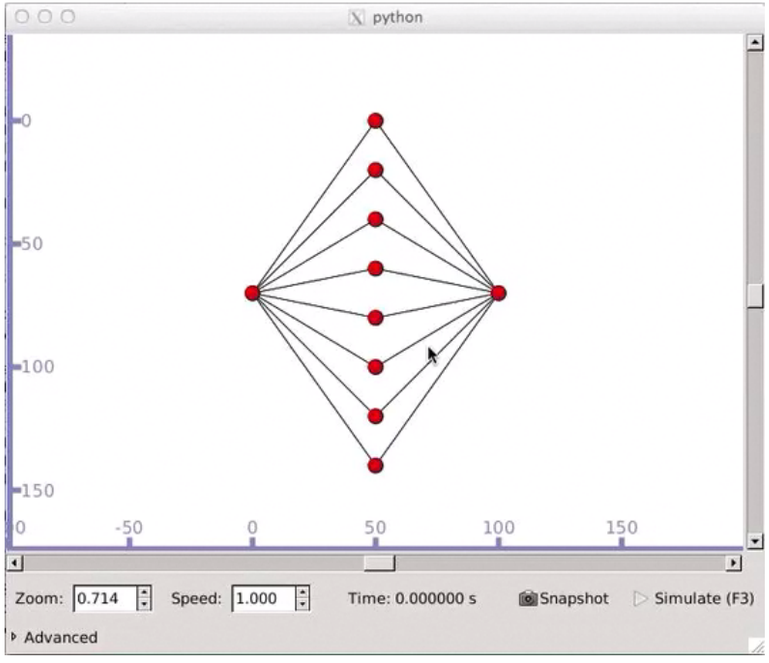
- LKL システムコール**のみ**を呼びだす標準ライブラリ
- カーネルに対応して、(仮想的な) CPU アーキテクチャ拡張
- プログラムは、通常の標準ライブラリとしてリンクする
  - システムコールを経由して直接ホストの資源アクセスは不可
- musl libc の拡張として実装



使い方 (アプリケーション)

- Use Case 1: ネットワークシミュレータとの結合
- Use Case 2: 簡易カーネルバイパス
- Use Case 3: カーネルコードを再利用したアプリケーション開発
- Use Case 4: Unikernel

Use Case 1: ネットワークシミュレータとの結合

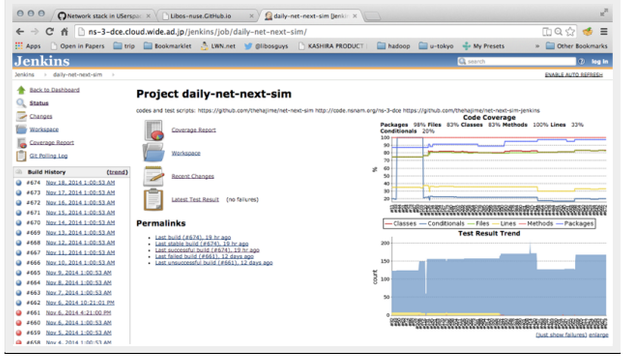


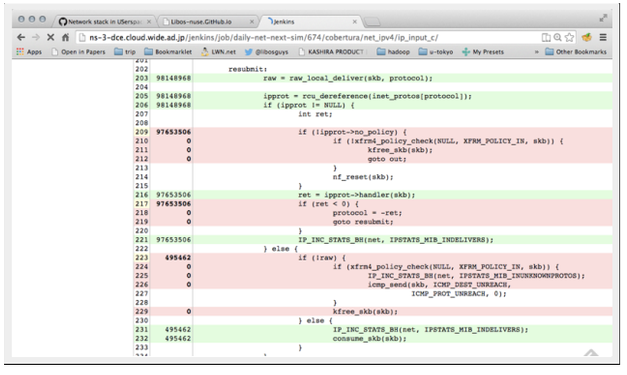
Linux Multipath-TCP の実験可視化

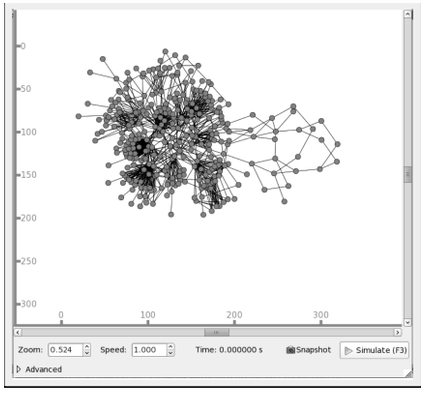
- ネットワーク状態の調査
- ns-3 ネットワークシミュレータ
- 多種のモデル
  - NIC
  - ノード移動
  - トラフィック
  - タイミング
- 単一プロセスにて複数ノード動作
  - dlmopen(3) にて実現 (シンボル退避)
  - システムコール再実装 (ノード振り分け)
- **再現性 100 %** (実験・バグの再現)

Use Case 1: ネットワークシミュレータとの結合

- カーネルのネットワークスタックテストツールとしても利用
  - Regression テスト (を複雑なネットワークで)
  - 再現率 100 % (仮想クロックを利用)
  - コードカバレッジ測定 (シミュレータの疑似乱数利用)
  - Valgrind でデバッグ







```
==5864== Memcheck, a memory error detector
==5864== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==5864== Using Valgrind-3.6.0.SVN and LibVEX; rerun with -h for copyright info
==5864== Command: ../build/bin/ns3test-dce-vdl --verbose
==5864==
==5864== Conditional jump or move depends on uninitialised value(s)
==5864==    at 0x7D5AE32: tcp_parse_options (tcp_input.c:3782)
==5864==    by 0x7D65DCB: tcp_check_req (tcp_minisocks.c:532)
==5864==    by 0x7D63B09: tcp_v4_hnd_req (tcp_ipv4.c:1496)
==5864==    by 0x7D63CB4: tcp_v4_do_rcv (tcp_ipv4.c:1576)
==5864==    by 0x7D6439C: tcp_v4_rcv (tcp_ipv4.c:1696)
==5864==    by 0x7D447CC: ip_local_deliver_finish (ip_input.c:226)
==5864==    by 0x7D442E4: ip_rcv_finish (dst.h:318)
==5864==    by 0x7D2313F: process_backlog (dev.c:3368)
==5864==    by 0x7D23455: net_rx_action (dev.c:3526)
==5864==    by 0x7CF2477: do_softirq (softirq.c:65)
==5864==    by 0x7CF2544: softirq_task_function (softirq.c:21)
==5864==    by 0x4FA2BE1: ns3::TaskManager::Trampoline(void*) (task-manager.cc:261)
==5864== Uninitialised value was created by a stack allocation
==5864==    at 0x7D65B30: tcp_check_req (tcp_minisocks.c:522)
==5864==
```

Use Case 2: 簡易カーネルバイパス

- LD_PRELOAD により、実行時にシステムコール迂回
- 置きかえ可能なシステムコール (シンボル) に制限あり
- LKL と外界 (ホストOS)の両方のシステムコールを利用可能

ホストカーネルに影響なく新機能を導入可能

```
LD_PRELOAD=liblkl-super-tcp++.so firefox
```

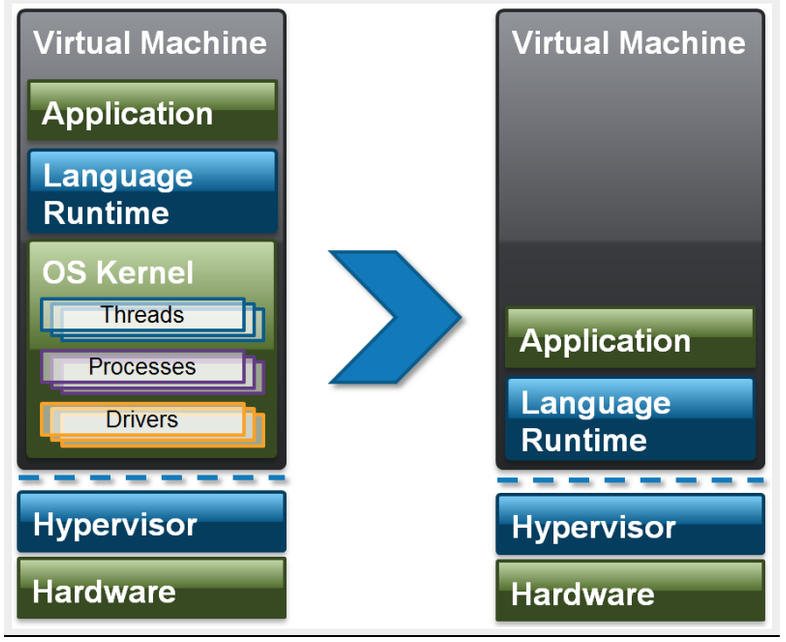
Use Case 3: カーネルコードを再利用したアプリケーション開発

- カーネルのコードを**移植なし**にユーザ空間でライブラリとして利用
- LKL と外界 (ホストOS)の両方のシステムコールを利用可能

- カーネル実装を利用したファイルシステムアクセス
- 例： ext4 フォーマットのディスクイメージにアクセス
  1. ディスクイメージを開く (*CreateFile()*)
  2. lkl_sys_mount()
  3. lkl_sys_read()
  4. ホストの別ファイルへ書きこみ (*WriteFile()*)

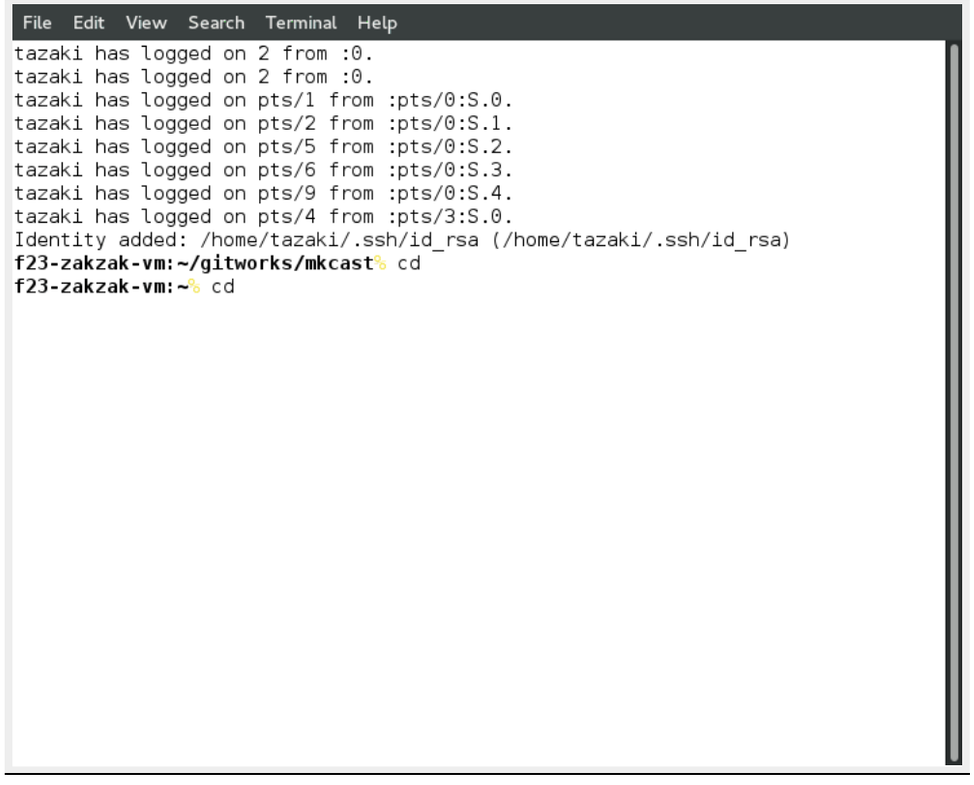
Use Case 4: Unikernel

- 単一アプリケーションにリンクさせる LKL
  - python + LKL, nginx + LKL
- LKL システムコールのみ利用可能
  - musl libc 移植
- rump ハイパーコール利用 (frankenlibc)
  - OS のないホスト上でも動作 (rumprun)
  - (on Xen Mini-OS)
- Work in progress

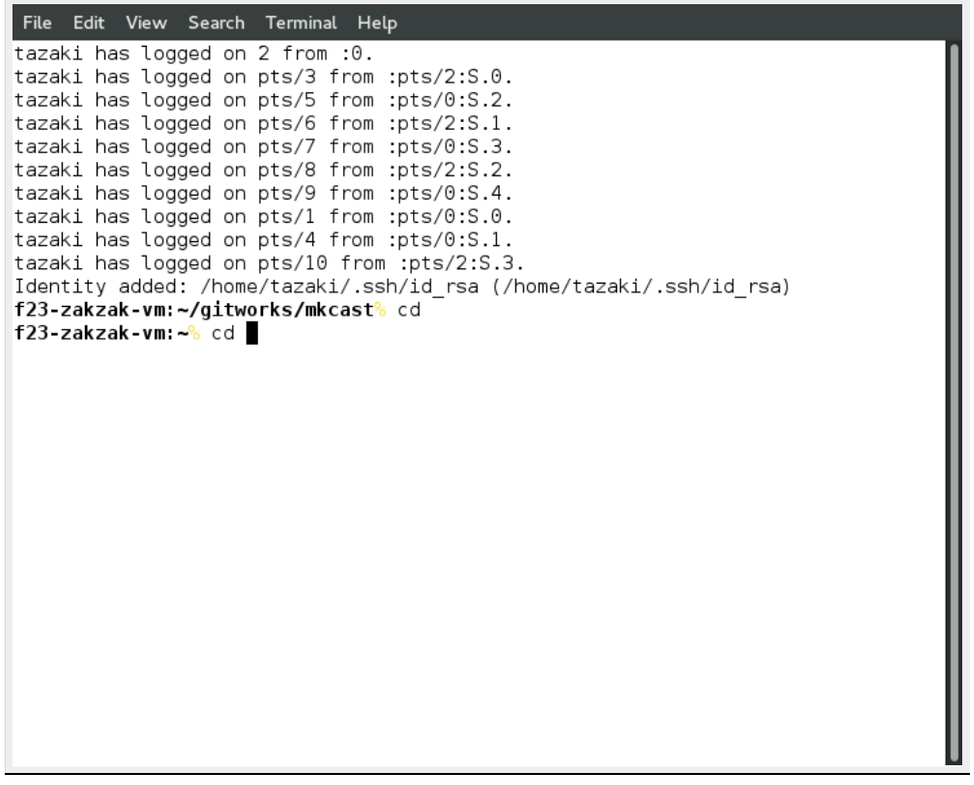


- http://www.linux.com/news/enterprise/cloud-computing/751156-are-cloud-operating-systems-the-next-big-thing-

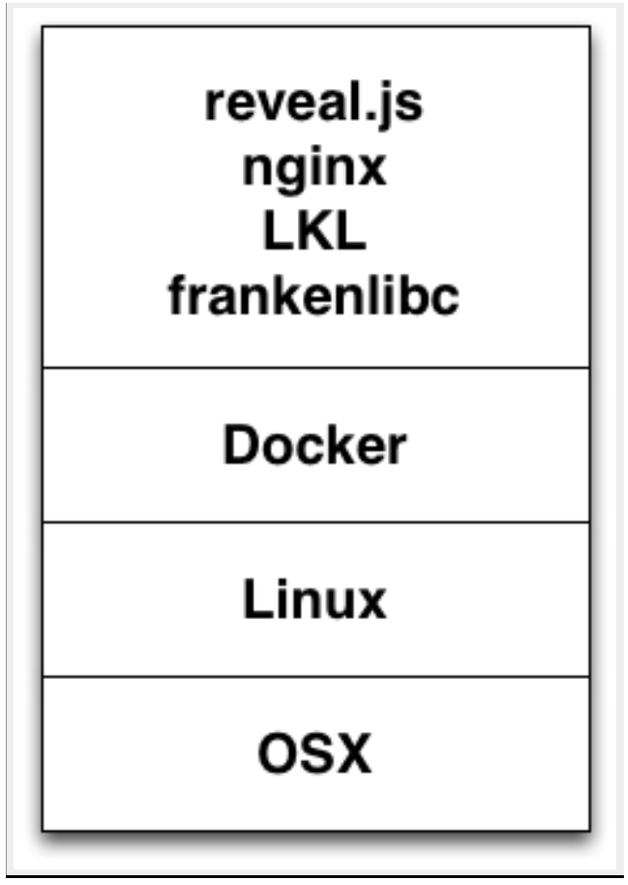
demos with linux kernel library



Unikernel on Linux (ping6 command embedded kernel library)



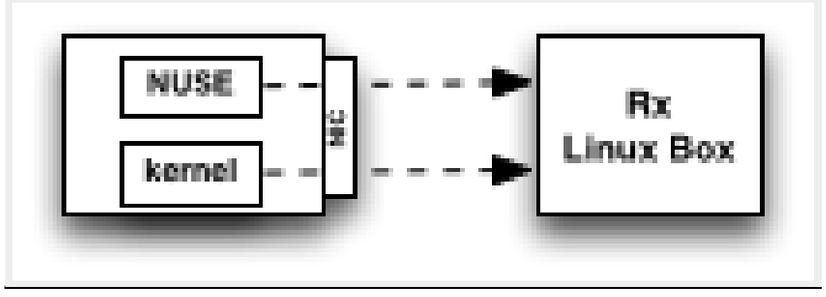
Unikernel on qemu-arm (hello world)

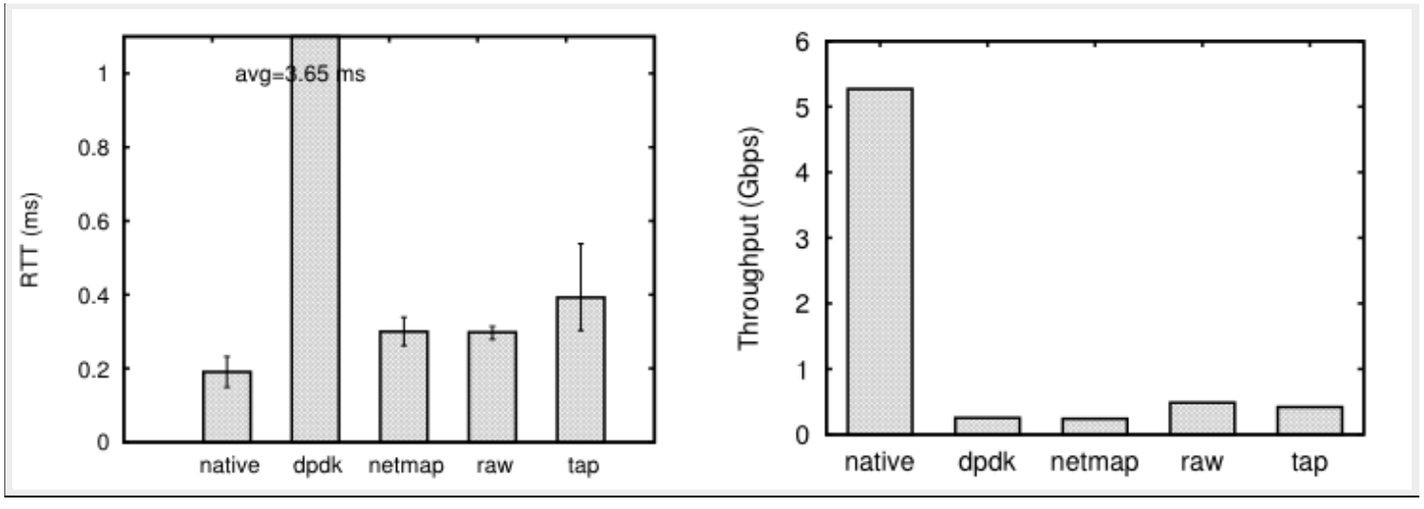


nginx (linked) LKL running on Linux

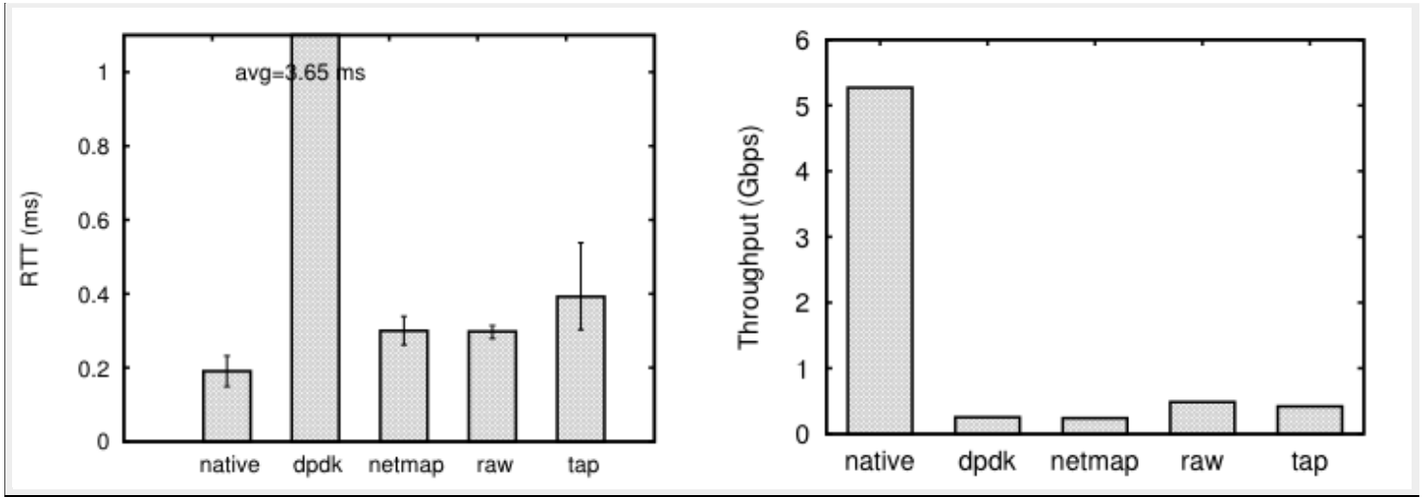
ベンチマーク

- 10Gbps Ethernet リンク (LibOS, 素の Linux)
- 8コア CPU
- ネットワークバックエンド (DPDK, netmap, rawソケットなど)
- RTT と Throughput 測定 (UDP 1024 byte パケット)





ベンチマーク (cont'd)



- 単純にカーネル迂回しただけでは速くはならない
- 原因
  - キャッシュミス
  - 多量の(ホスト)システムコールの数

ボトルネック分析

- ネットワークスタック部
  - CPU コアローカルではないプロセス内通信
  - 1パケット毎の処理多数 (バルク処理)
  - システムコールのオーバヘッド (ホストバックエンド実装)
- LKL 本体
  - 迂回 (Indirection)の影響

	性能	既存資産	最新版追従
full scratch	++	--	+
porting	+	+/-	--
anykernel	-	++	++

	性能	既存資産	最新版追従
full scratch	++	--	+
porting	+	+/-	--
anykernel	+	++	++

まとめ

- カーネルのコードは再利用するものです!
  - 20 年近くの資産を捨てるのは無駄
  - (面倒な)移植なしに
  - ライブラリとして (ユーザ空間プログラムに限らず)
  - 必要な部分のみ利用可能
- オープンソースプロジェクトとして開発
  - https://github.com/lkl/linux

参考資料

- Linux Kernel Library
  - Purdila et al., LKL: The Linux kernel library, RoEduNet 2010.
  - https://github.com/lkl/linux
- Rumpkernel (dissertation)
  - Kantee, Flexible Operating System Internals: The Design and Implementation of the Anykernel and Rump Kernels, Ph.D Thesis, 2012
- Linux LibOS 一般
  - Tazaki et al. Direct Code Execution: Revisiting Library OS Architecture for Reproducible Network Experiments, CoNEXT 2013
  - http://libos-nuse.github.io/ (LibOS in general)
  - https://lwn.net/Articles/637658/