スライド: https://pibvt.net/kernelvm-2025-0511.pdf

Mewz on libkrun

松本直樹(@PiBVT)

2025/05/11

Kernel/VM探検隊@関西 11回目

スライド: https://pibvt.net/kernelvm-2025-0511.pdf

https://github.com/mewz-project/mewz

https://github.com/mewz-project/wasker

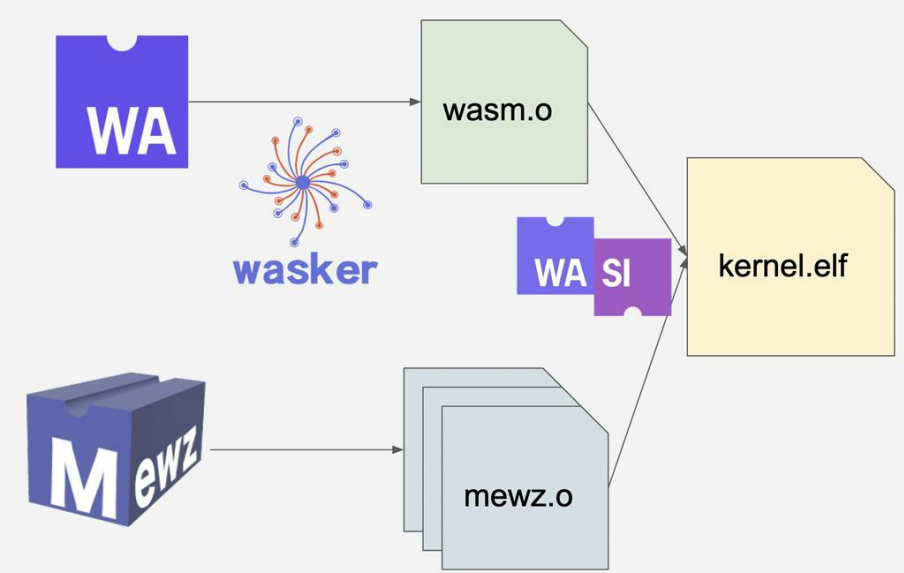
Mewz, Wasker について

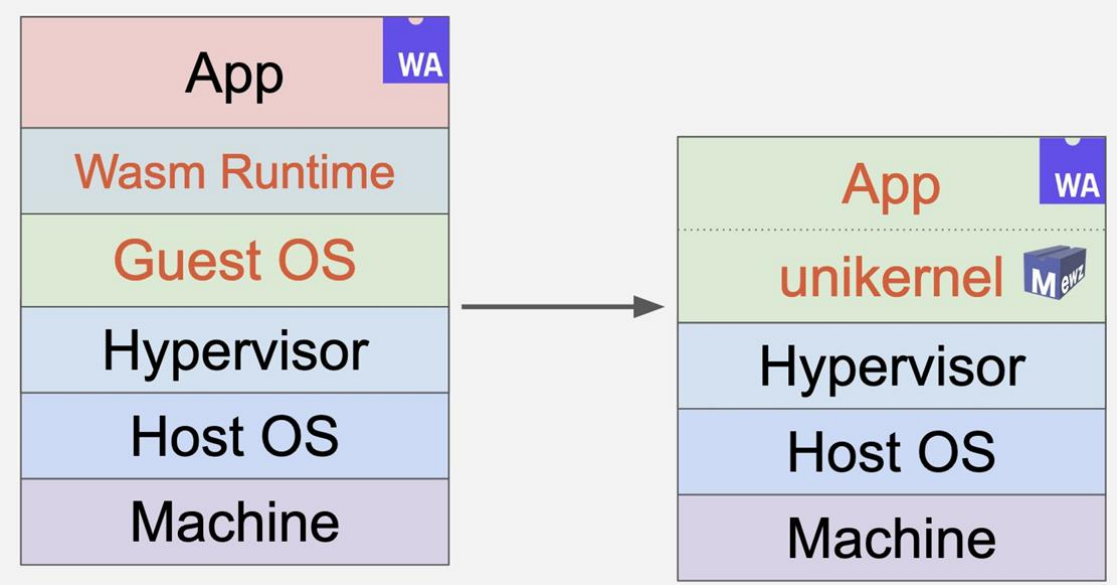
Wasker: WebAssembly モジュールを**オブジェクトファイル**に変換する AoT コンパイラ

- WASI の呼び出しを未解決シンボルとして変換する = リンク先は WASI 関数を実装すればよい

Mewz: ↑のオブジェクトファイルをリンクし動作する unikernel

- シンプルな kernel にアプリもまとめてリンクして動作する



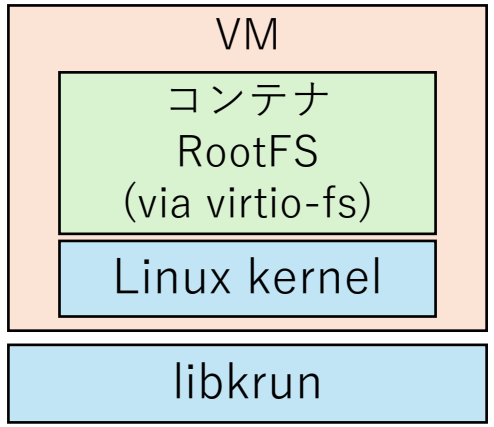


https://www.ipa.go.jp/jinzai/mitou/it/2023/m42obm000000azzl-att/seikagaiyou-sg-1.pdf より

スライド: https://pibvt.net/kernelvm-2025-0511.pdf

libkrun について

https://github.com/containers/libkrun



Sergio López 氏(@RedHat) が開発する軽量 VMM

- コンテナを VM として動作させるために開発されている
- Rust で実装
- デバイスのサポートは必要最低限(virtio 系)
- コンテナの RootFS を virtio-fs 経由でマウントし動作
- ネットワーク周りは**意図的に**ホストと共有
  → ホスト側の network namespace による分離(Pod としても利用可能)
  - passt: virtio-net 経由でやり取りされる L2 パケットをホスト側の L4 なソケット操作に置換
    QEMU の user-mode networking 相当
  - tsi: ゲスト内のソケット操作を virtio-vsock 経由でホストに中継(詳細は後述)

参考

- https://logmi.jp/main/technology/324735
- https://rheb.hatenablog.com/entry/libkrun-intro

スライド: https://pibvt.net/kernelvm-2025-0511.pdf

libkrun の概観

https://pibvt.hateblo.jp/entry/2024/12/30/213756

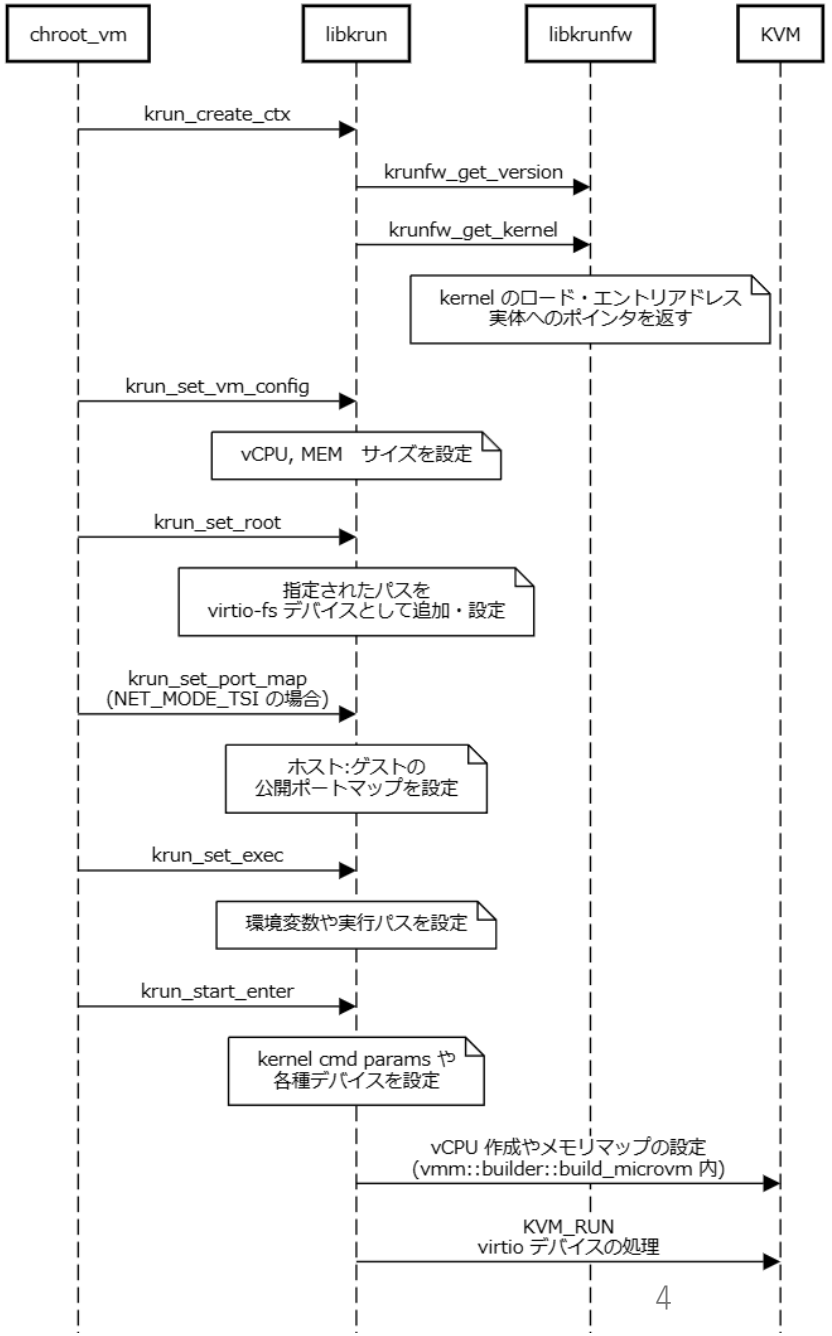
“lib”krun = VMM 実体は共有ライブラリとして提供

- krun_* API を提供
- Linux kernel も共有ライブラリ(libkrunfw)として提供

```c
// Create the configuration context.
ctx_id = krun_create_ctx();
if (ctx_id < 0) {
    errno = -ctx_id;
    perror("Error creating configuration context");
    return -1;
}

// Configure the number of vCPUs (1) and the amount of RAM (512 MiB).
if (err = krun_set_vm_config(ctx_id, 4, 4096)) {
    errno = -err;
    perror("Error configuring the number of vCPUs and/or the amount of RAM");
    return -1;
}
```

https://github.com/containers/libkrun/blob/main/examples/chroot_vm.c より



スライド: https://pibvt.net/kernelvm-2025-0511.pdf

Mewz on libkrun までの道のり

「Mewz と libkrun、軽量なもの同士相性よさそう 💡 」
→ Mewz on likbrun をやってみることに

本発表は**ブログ記事「Mewz on libkrun」シリーズ**に関する発表です
ブログ記事: https://pibvt.hateblo.jp/entry/2024/12/30/213724

Mewz を libkrun で動かすまでの道のり

1. Linux zeropage への対応 (ここで最小限のブートが可能になる)
2. kernel command-line params への対応
3. Virtio Over MMIO への対応(virtio-net, virtio-console)

スライド: https://pibvt.net/kernelvm-2025-0511.pdf

Linux zeropage

https://pibvt.hateblo.jp/entry/2024/12/30/213824

Linux zeropage: メモリ領域等各種パラメータを保持する (libkrun が情報を埋め込む)

参考: https://github.com/torvalds/linux/blob/01f95500a162fca88cefab9ed64ceded5afabc12/arch/x86/include/uapi/asm/bootparam.h#L115-L163

※ Mewz は multiboot protocol のみ対応

今回は E820 (利用可能なメモリ領域) についてのみ対応すれば OK

1. zeropage からE820 エントリ数 (@ 0x7000 + 0x1E8, u8) を読み取る
2. E820 のエントリ(@ 0x7000 + 0x2D0 + 20 * n)を順番に読み取る
3. type == 1 (E820 RAM) な領域を拾い集める
4. Mewz 側で利用可能な領域としてメモリ管理を初期化

```
pub const E820Entry = extern struct {
    addr: u64 align(4),
    size: u64 align(4),
    type_: u32 align(4),
};
```

multiboot との共存(QEMU での動作)は magic value で行う

```
if (boot_magic == 0x2badb002) {
    log.info.print("booted with multiboot1\n");
    const bootinfo = @as(*multiboot.BootInfo, @ptrFromInt(boot_params));
    printBootinfo(boot_magic, bootinfo);
    mem.init(bootinfo);
    param.parseFromArgs(util.getString(bootinfo.cmdline));
} else {
    log.info.print("booted with linux zero page\n");
    const info = zeropage.parseZeroPageInfo(0x7000);
    mem.initWithZeroPage(info);
}
```

スライド: https://pibvt.net/kernelvm-2025-0511.pdf

Kernel command-line params

https://pibvt.hateblo.jp/entry/2024/12/30/213824

libkrun は Virtio MMIO デバイスの情報を kernel params として提供

※ libkrun は Virtio PCI を利用しないためスキャンによる検出はできない(後述)

zeropage から kernel params へのアドレスを取得し↓の文字列をパースする

```
reboot=k panic=-1 panic_print=0 nomodule console=hvc0 rootfstype=virtiofs rw quiet no-kvmapf …
```

パースの手順

1. 空白区切りで取り出す
2. ネットワーク情報, Virtio MMIO デバイス関連以外はとりあえず無視
   - ネットワーク(`ip=192.168.10.2/24`, `gateway=192.168.10.1`)
   - Virtio デバイス(`virtio_mmio.device=4K@0xd0004000:9`) サイズ@アドレス:IRQLine
3. 取得した情報で各種設定を行う

スライド: https://pibvt.net/kernelvm-2025-0511.pdf

Virtio Over MMIO

https://pibvt.hateblo.jp/entry/2024/12/30/213842

Kernel params の情報をもとに Virtio MMIO なデバイスを初期化する

1. アドレスに対して→をマップして情報取得
2. ↓にあるようにVirtqueue の初期化を行う
3. 残りの初期化は Virtio Over PCI なデバイスと同じ手順で行う

4.2.3 MMIO-specific Initialization And Device Operation

4.2.3.1 Device Initialization

4.2.3.1.1 Driver Requirements: Device Initialization

The driver MUST start the device initialization by reading and checking values from *MagicValue* and *Version*. If both values are valid, it MUST read *DeviceID* and if its value is zero (0x0) MUST abort initialization and MUST NOT access any other register.
Drivers not expecting shared memory MUST NOT use the shared memory registers.

Further initialization MUST follow the procedure described in 3.1 Device Initialization.

4.2.3.2 Virtqueue Configuration

The driver will typically initialize the virtqueue in the following way:

1. Select the queue by writing its index to *QueueSel*.
2. Check if the queue is not already in use: read *QueueReady*, and expect a returned value of zero (0x0).
3. Read maximum queue size (number of elements) from *QueueSizeMax*. If the returned value is zero (0x0) the queue is not available.
4. Allocate and zero the queue memory, making sure the memory is physically contiguous.
5. Notify the device about the queue size by writing the size to *QueueSize*.
6. Write physical addresses of the queue's Descriptor Area, Driver Area and Device Area to (respectively) the *QueueDescLow*/*QueueDescHigh*, *QueueDriverLow*/*QueueDriverHigh* and *QueueDeviceLow*/*QueueDeviceHigh* register pairs.
7. Write 0x1 to *QueueReady*.

4.2.2 MMIO Device Register Layout

MMIO virtio devices provide a set of memory mapped control registers followed by a device-specific configuration space, described in the table 4.1.

All register values are organized as Little Endian.

Table 4.1: MMIO Device Register Layout

Name Offset from base Direction	Function Description
MagicValue 0x000 R	Magic value 0x74726976 (a Little Endian equivalent of the "virt" string).
Version 0x004 R	Device version number 0x2. Note: Legacy devices (see 4.2.4 Legacy interface) used 0x1.
DeviceID 0x008 R	Virtio Subsystem Device ID See 5 Device Types for possible values. Value zero (0x0) is used to define a system memory map with placeholder devices at static, well known addresses, assigning functions to them depending on user's needs.
VendorID 0x00c R	Virtio Subsystem Vendor ID
DeviceFeatures 0x010 R	Flags representing features the device supports Reading from this register returns 32 consecutive flag bits, the least significant bit depending on the last value written to *DeviceFeaturesSel*. Access to this register returns bits *DeviceFeaturesSel* ∗ 32 to (*DeviceFeaturesSel* ∗ 32) + 31, eg. feature bits 0 to 31 if *DeviceFeaturesSel* is set to 0 and features bits 32 to 63 if *DeviceFeaturesSel* is set to 1. Also see 2.2 Feature Bits.
DeviceFeaturesSel 0x014 W	Device (host) features word selection. Writing to this register selects a set of 32 device feature bits accessible by reading from *DeviceFeatures*.
DriverFeatures 0x020 W	Flags representing device features understood and activated by the driver Writing to this register sets 32 consecutive flag bits, the least significant bit depending on the last value written to *DriverFeaturesSel*. Access to this register sets bits *DriverFeaturesSel* ∗ 32 to (*DriverFeaturesSel* ∗ 32) + 31, eg. feature bits 0 to 31 if *DriverFeaturesSel* is set to 0 and features bits 32 to 63 if *DriverFeaturesSel* is set to 1. Also see 2.2 Feature Bits.
DriverFeaturesSel 0x024 W	Activated (guest) features word selection Writing to this register selects a set of 32 activated feature bits accessible by writing to *DriverFeatures*.
QueueSel 0x030 W	Virtqueue index Writing to this register selects the virtqueue that the following operations on *QueueSizeMax*, *QueueSize*, *QueueReady*, *QueueDescLow*, *QueueDescHigh*, *QueueDriverLow*, *QueueDriverHigh*, *QueueDeviceLow*, *QueueDeviceHigh* and *QueueReset* apply to.
QueueSizeMax	Maximum virtqueue size

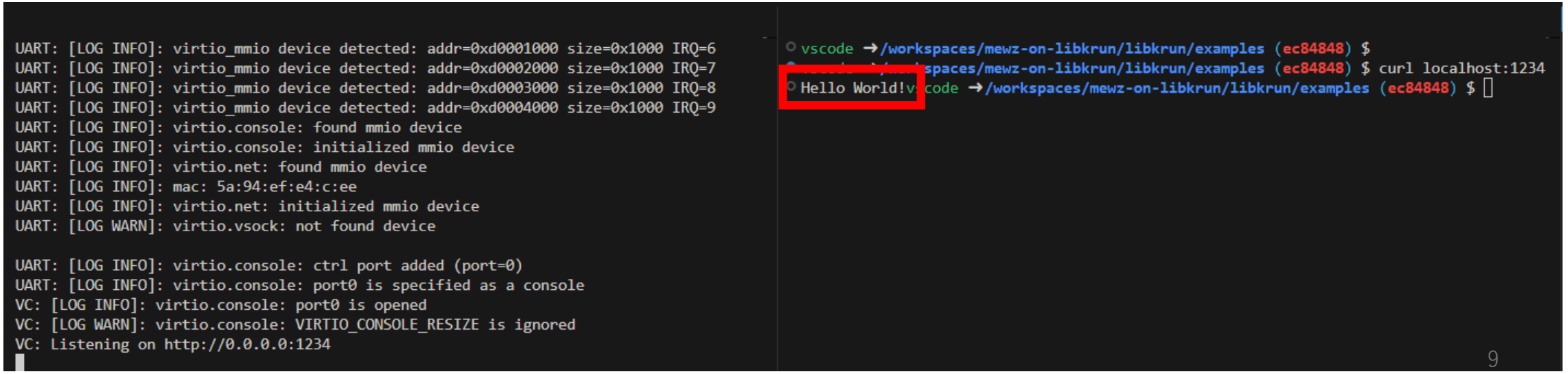
https://docs.oasis-open.org/virtio/virtio/v1.3/csd01/virtio-v1.3-csd01.html#x1-1820002 より

スライド: https://pibvt.net/kernelvm-2025-0511.pdf

動作

現状の実装を GitHub にて公開中

```
$ sudo apt install -y passt
$ git clone --recursive http://github.com/naoki9911/mewz-on-libkrun
$ cd mewz-on-libkrun
$ docker run --rm -v $(pwd):/work ghcr.io/naoki9911/mewz-on-libkrun:main /work/build.sh
$ cd likbrun/examples
$ sudo LD_LIBRARY_PATH=../lib ./chroot_vm --net=passt dummy dummy
```



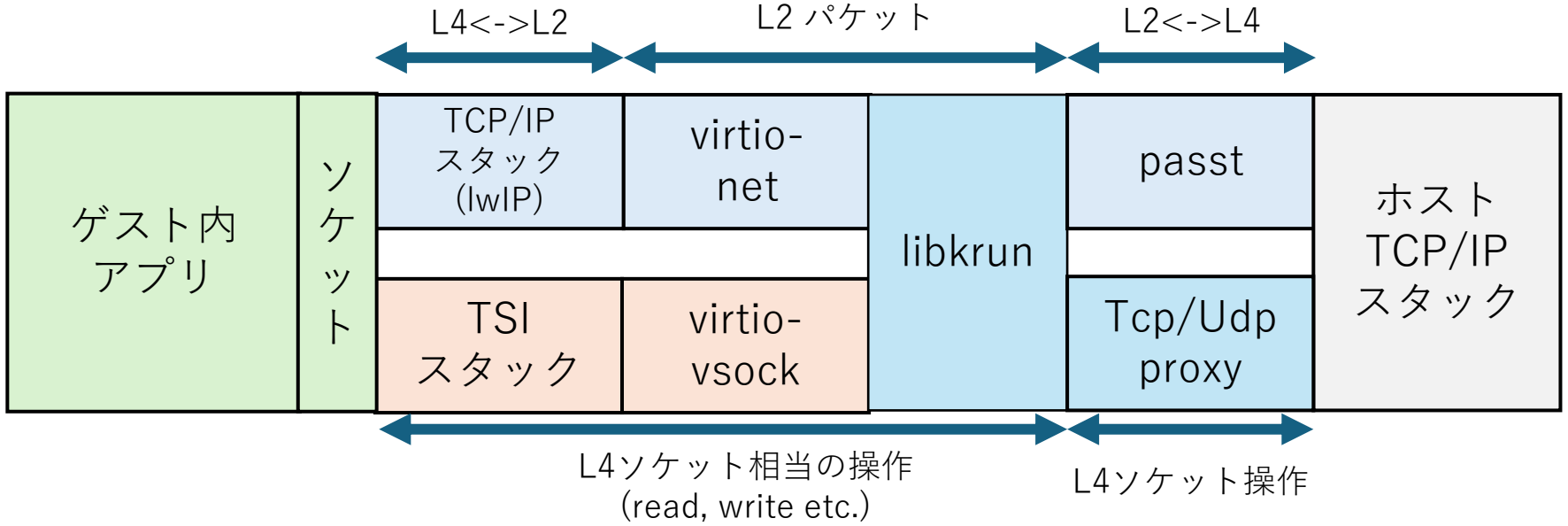
スライド: https://pibvt.net/kernelvm-2025-0511.pdf

発展: Transparent Socket Impersonation

Transparent Socket Impersonation(TSI):

ゲストはTCP/IP スタックを持たず、virtio-vsock 経由でホストの TCP/IP スタックを利用する

- passt のようなL2<->L4 ソケット操作の変換が存在しないためシンプル
- ゲストが TCP/IP スタックを持たなくてよい = シンプル



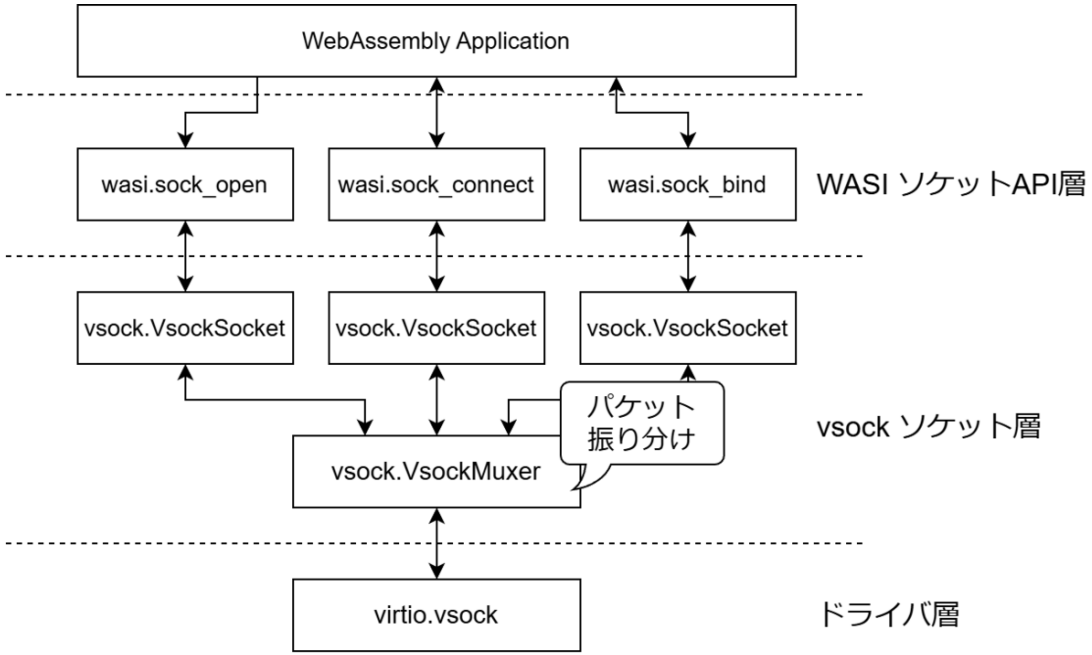
スライド: https://pibvt.net/kernelvm-2025-0511.pdf

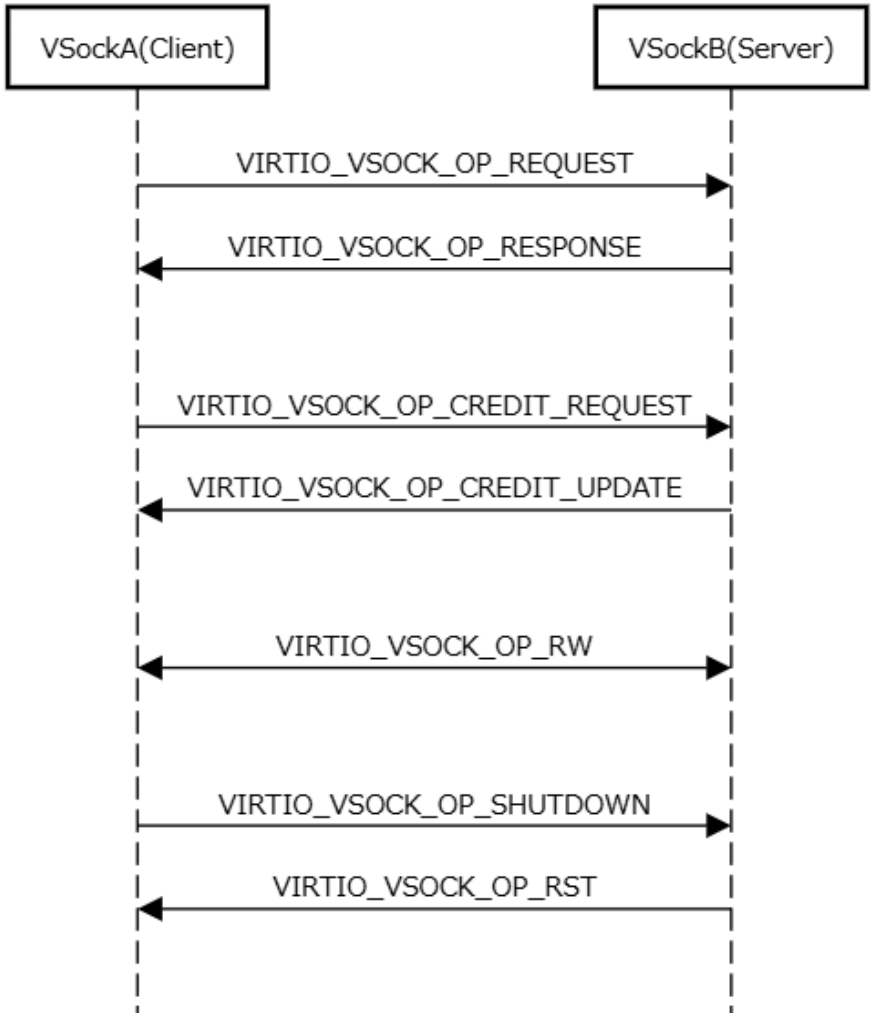
TSI の実装: virtio-vsock

https://pibvt.hateblo.jp/entry/2025/01/07/235839
https://pibvt.hateblo.jp/entry/2025/01/13/004833

TSI には virtio-vsock の実装が必要 → 実装した

- ソケットに対応したパケットの振り分け機構
- 各ソケットにおけるハンドシェイク(→の流れ)
- WASI (wasmedge_wasi_socket) にも対応
  ＝ WASM アプリからも AF_VSOCK を叩ける





スライド: https://pibvt.net/kernelvm-2025-0511.pdf

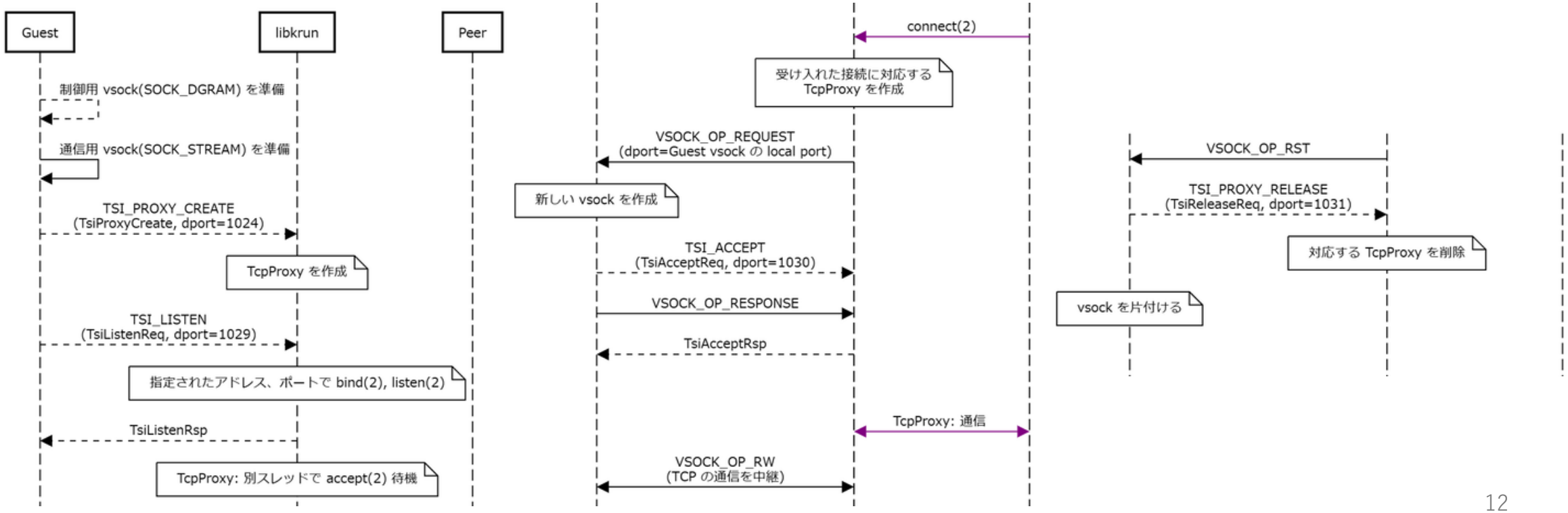
TSI の実装

https://pibvt.hateblo.jp/entry/2025/01/18/195434

TCP/IP スタック -> TSI への置き換えを実装

※本来は kernel param で “**tsi**” が指定されるためそれに応じて切り替えるべき

libkrun 側の TSI モジュールと通信して **bind, listen** 等を処理



スライド: https://pibvt.net/kernelvm-2025-0511.pdf

TSI の動作

TCP/IP スタック(lwIP)をリンクせずとも HTTP サーバーが動く

```
kernel.linker_script = b.path("src/x64.ld");
kernel.addAssemblyFile(b.path("src/boot.S"));
kernel.addAssemblyFile(b.path("src/interrupt.S"));
kernel.addObjectFile(b.path("build/newlib/libc.a"));
//kernel.addObjectFile(b.path("build/lwip/libtcpip.a"));
//kernel.addObjectFile(b.path("build/lwip/liblwipcore.a"));
//kernel.addObjectFile(b.path("build/lwip/liblwipallapps.a"));
kernel.addCSourceFile(.{ .file: LazyPath = b.path("src/c/newlib_support.c"), .flags: []const []const u8 = &.{ "-I", "submodules/newlib/newlib/libc/include" } });
//kernel.addCSourceFile(.{ .file = b.path("src/c/lwip_support.c"), .flags = &.{ "-I", "submodules/newlib/newlib/libc/include" } });
```

```
[LOG INFO]: virtio_mmio device detected: addr=0xd0000000 size=0x1000 IRQ=5
[LOG INFO]: virtio_mmio device detected: addr=0xd0001000 size=0x1000 IRQ=6
[LOG INFO]: virtio_mmio device detected: addr=0xd0002000 size=0x1000 IRQ=7
[LOG INFO]: virtio_mmio device detected: addr=0xd0003000 size=0x1000 IRQ=8
[LOG INFO]: virtio_mmio device detected: addr=0xd0004000 size=0x1000 IRQ=9
[LOG INFO]: virtio.console: found mmio device
[LOG INFO]: virtio.console: ctrl port added (port=0)
[LOG INFO]: virtio.console: initialized mmio device
[LOG INFO]: virtio.vsock: found mmio device
[LOG INFO]: virtio.vsock: guest CID is 3
[LOG INFO]: virtio.vsock: initialized mmio device

[LOG INFO]: virtio.console: port0 is specified as a console
[LOG INFO]: virtio.console: port0 is opened
[LOG WARN]: virtio.console: VIRTIO_CONSOLE_RESIZE is ignored
Listening on http://0.0.0.0:1234
```

```
● vscode ➜/workspaces/mewz-on-libkrun/mewz (mewz-on-libkrun) $ curl localhost:1234
○ Hello World!vscode ➜/workspaces/mewz-on-libkrun/mewz (mewz-on-libkrun) $
```

スライド: https://pibvt.net/kernelvm-2025-0511.pdf

まとめ

Mewz on libkrun

- libkrun を用いれば軽量に Mewz を動作させることが可能
- libkrun での動作には Linux zeropage, Virtio MMIO 等への対応が必要
- TSI を用いればゲスト側で TCP/IP スタックを持つ必要がなくなる

さらに詳しい内容

- ブログ記事: https://pibvt.hateblo.jp/entry/2024/12/30/213724
- リポジトリ: https://github.com/naoki9911/mewz-on-libkrun
- 蛇足: Intel64 と AMD64 の PML4 Paging 仕様の差異
  https://x.com/PiBVT/status/1755141978940211595