PCIデバイスの位置を誤魔化すハイパーバイザを作った

鬼頭 太郎 (petitstrawberry)
産業技術総合研究所

2026/05/30 Kernel/VM探検隊@関西 12回目

本研究は、JST CREST 21K17727 の支援を受けたものです。

はじめに

突然ですが...!

PCIデバイスが生えている場所 (BDF) を,
全く別の場所に見せかけたいと思ったことはありますか？

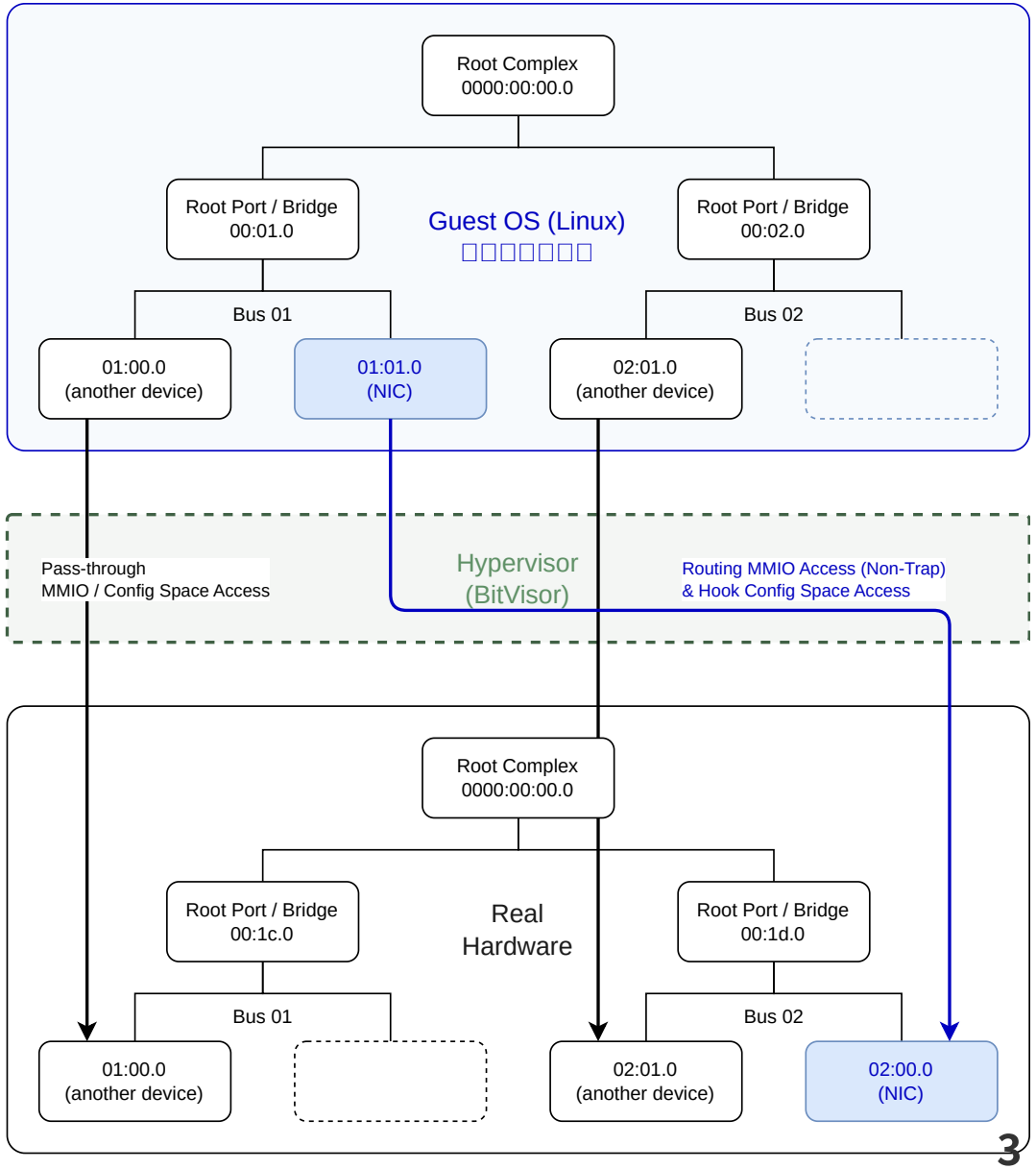
私はあります。(諸事情により)

ので、やりました。

- BDF (Bus/Device/Function)
  - [bus]:[device].[function] で表されるPCIデバイスを一意に識別するための表記
  - `lspci` コマンドなどで見られるやつ
  - ```
    ❯ lspci
    00:00.0 Host bridge: Intel Corporation Raptor Lake-S Host Bridge/DRAM Controller (rev 01)
    00:02.0 VGA compatible controller: Intel Corporation Raptor Lake-S GT1 [UHD Graphics 770] (rev 04)
    00:06.0 PCI bridge: Intel Corporation Raptor Lake PCI Express 4.0 Graphics Port (rev 01)
    ```
    ```

完成イメージ

- 任意のPCIデバイスを任意のBDFに移動させる
  - NIC(02:00.0)がゲストLinuxからは
    別の場所(01:01.0)存在するように見せかける
  - もちろん, 通常通り機能するようにする
  - デバイス操作などは基本的にパススルー
  - ただ, BDFを偽装する**だけ**
  - 元デバイスを**元のBDFには見えないようにする**
- PCI Bridgeなどを含めたトポロジ全体の
  エミュレーションはせず, 基本的にはホストの
  構成をそのまま使う
- 薄い仮想化のためにBitVisor[1]を用いる



[1]: BitVisor, https://www.bitvisor.org/ja/

デバイスのBDFを偽装する方法

今回はBDF_X(02:00.0)に存在するNICをBDF_Y(01:01.0)に移動させることを目標にする

達成すべきことは大きく分けると2つになる

- LinuxのPCI enumeration を騙して, BDF_Xの元デバイスを隠し, BDF_Yにデバイスを見せる
- 実際のデバイスのBDF_X $\neq$ ゲストに見せるBDF_Y で生じる不整合を吸収する
  - BARを誤魔化す / MMIOをねじ曲げる
  - 電源管理問題

PCI enumeration を騙す

PCIデバイスの列挙はPCI Configuration Spaceへの読み書きから始まる

Linux: 各Bus/Device/Functionに対してConfig空間を読む

1. Vendor ID / Device ID を読む
    1. 0xFFFFなら「デバイスなし」と判断して次へ
2. Header Type / Class Code / BAR などを読む
3. BARに0xFFFFFFFFを書いて, 必要なMMIO/IOサイズを調べる
4. 空いているアドレスをBARに書き戻す
5. Command RegisterでMemory Space / Bus Masterなどを有効化

Config空間の値を誤魔化して返す必要があるので, ここは仕方なくTrapしてHookする

- BDF_X（02:00.0）へのアクセスは全て失敗させて, デバイスなしのように見せる
- BDF_Y（01:01.0）へのアクセスは全てBDF_XのConfig空間を読んでいるように見せる

あかん...

```
[    2.684953] pci 0000:01:01.0: BAR 1 [mem 0x81400000-0x81400fff]: can't claim; address conflict with PCI Bus 0000:02 [mem 0x81400000-0x815fffff]
```

どうやら, 移動させたPCIデバイスのMMIO領域がBridge Window(Linuxが定める, Bridge/Busごとに利用可能なメモリ領域)の範囲を超えて不正である(他のBusと衝突している)と怒られている

- NICが存在するのは本来BDF_X(02:00.0)
- BIOS/UEFIにより, BitVisorやLinuxの起動前に, あらかじめ, Bus 02のBridgeの配下で利用可能な領域(0x81400000-0x815fffff) を設定
- LinuxはPCI enumerationでBARの値を読んで現状確認し, あらかじめ有効な値が設定されていたらそれを利用
- しかし, LinuxからNICはBus 02ではなくBus 01のデバイスであるように見えるので, 不整合発生, 怒られる

BDFだけを捻じ曲げてもダメみたい

BARを誤魔化す

こうなった以上, Linuxに怒られないアドレスをBARに設定するしかない

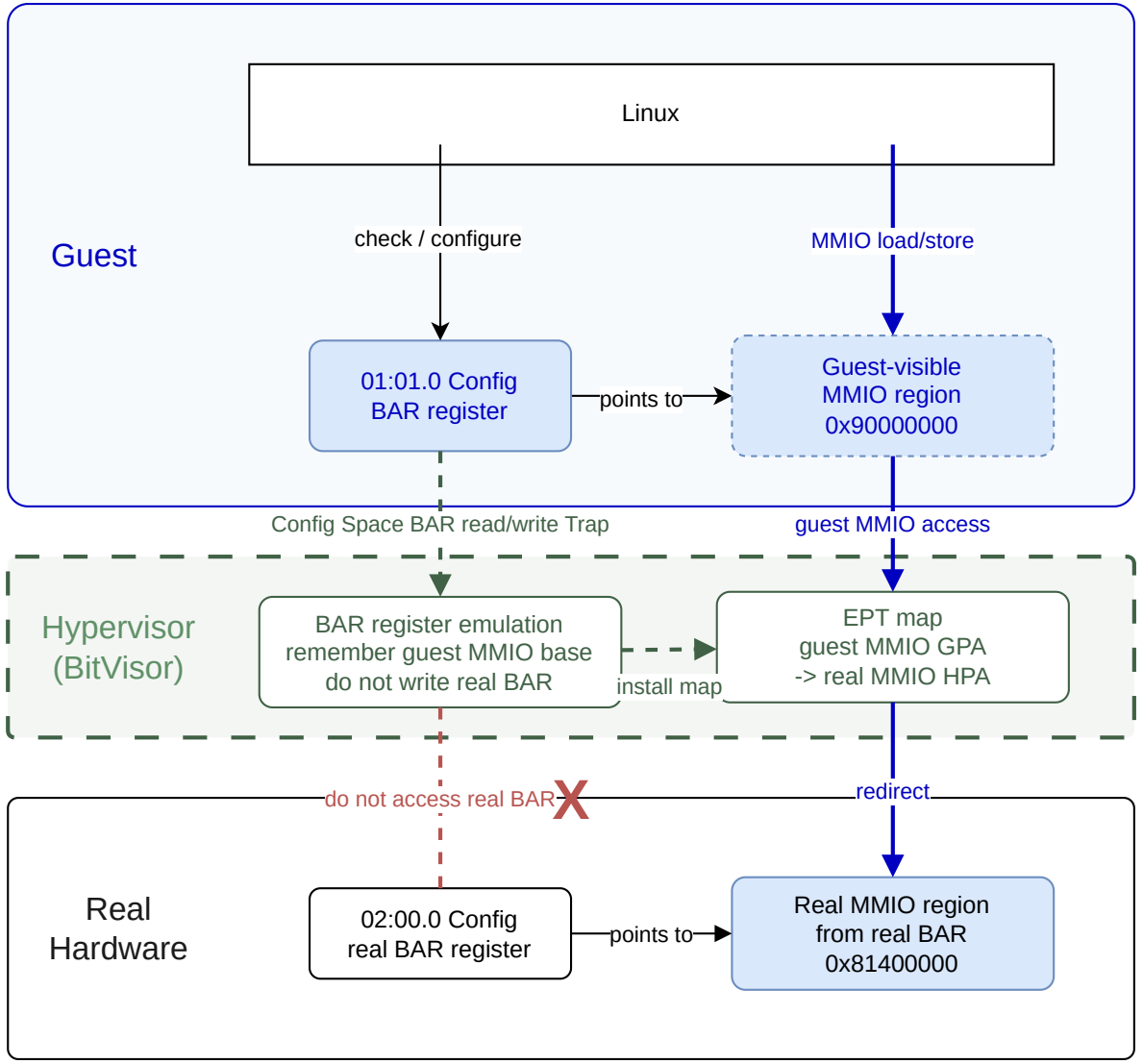
考えられる方法は2つ

- Linux起動前にBitVisor側で移動先BDF_Y(01:01.0)のBus 01で有効な範囲かつ, 他のデバイスで使われていなさそうな領域をセットしておく
- NICのBARが未初期化・不正な値っぽく見せることで, Bus 01で有効かつ, 他のデバイスが使っていない場所へ**Linux自身に再配置させる**

前者は, 時と場合によって変わったり, 構成変更に弱そう, 面倒くさそうなので後者を選択

MMIOアクセスをねじ曲げる

- BARへのアクセスをHookして,
  BAR操作をエミュレートする
  - 実際のデバイスのBARには書き込ませない
    (**実際のMMIO領域はそのまま固定**しておく)
  - Linuxが書き込んできたアドレスを覚えておく
- Linuxが書き込んできたアドレスから
  実デバイスのMMIO領域への**EPTマップ**を
  貼って, Non-Trapかつ,
  LinuxがMMIO可能にしておく



電源管理問題

やったか??? ダメでした

```
Unable to change power state from D3cold to D0
PCIe link lost
```

問題のMMIO領域の割当ては問題なく完了したが, デバイスの電源が落ちて, うんともすんとも言わない (NICのレジスタをREADしても全て `0xFF`)

どうやら、トポロジと実際の不整合はBridge経由の電源管理などもいろいろ影響が出そうな感じ... これ以上は少し辛そうなので, 強硬手段に出た

```
pcie_port_pm=off pcie_aspm=off pci=noaer
```

をbootargsに追加して, PCI系の電源管理周りを軒並み無効化

BAR/MMIOはEndpoint単体の問題として吸収できたが, 電源管理はRoot Port/Bridge/ACPIを含むトポロジ依存の問題だったため, BDF偽装だけでは破綻したと考えられる...
　BDF_Xには何もデバイスが存在しないようにLinuxには見えるため, このポートの電源を落とすが, 実デバイスがそこに存在しているため, 電源が落ちてしまう...
なので, 本来はBridgeのエミュレーションなども必要だったのかもしれないが, そこまでやると薄い仮想化ではなくなりそうなので, 強硬手段で解決した

こんな感じ

Before

```
ra25kito@xeon-e2376g:~$ lspci -tv
-[0000:00]-+-00.0  Intel Corporation Device 4c43
           +-01.0-[01]--+-00.0  NVIDIA Corporation GK208B [GeForce GT 730]
           |            \-00.1  NVIDIA Corporation GK208 HDMI/DP Audio Controller
           +-01.1-[02]--+-00.0  Intel Corporation Ethernet Controller X710 for 10GbE SFP+
           |            \-00.1  Intel Corporation Ethernet Controller X710 for 10GbE SFP+
           +-06.0-[03]----00.0  Samsung Electronics Co Ltd NVMe SSD Controller PM9A1/PM9A3/980PRO
```

After: NIC(X710)が02:00.0から01:01.0に移動しているのがわかる

```
-[0000:00]-+-00.0  Intel Corporation Device 4c43
           +-01.0-[01]--+-00.0  NVIDIA Corporation GK208B [GeForce GT 730]
           |            +-00.1  NVIDIA Corporation GK208 HDMI/DP Audio Controller
           |            \-01.0  Intel Corporation Ethernet Controller X710 for 10GbE SFP+
           +-01.1-[02]--
           +-06.0-[03]----00.0  Samsung Electronics Co Ltd NVMe SSD Controller PM9A1/PM9A3/980PRO
           +-14.0  Intel Corporation Tiger Lake-H USB 3.2 Gen 2x1 xHCI Host Controller
```

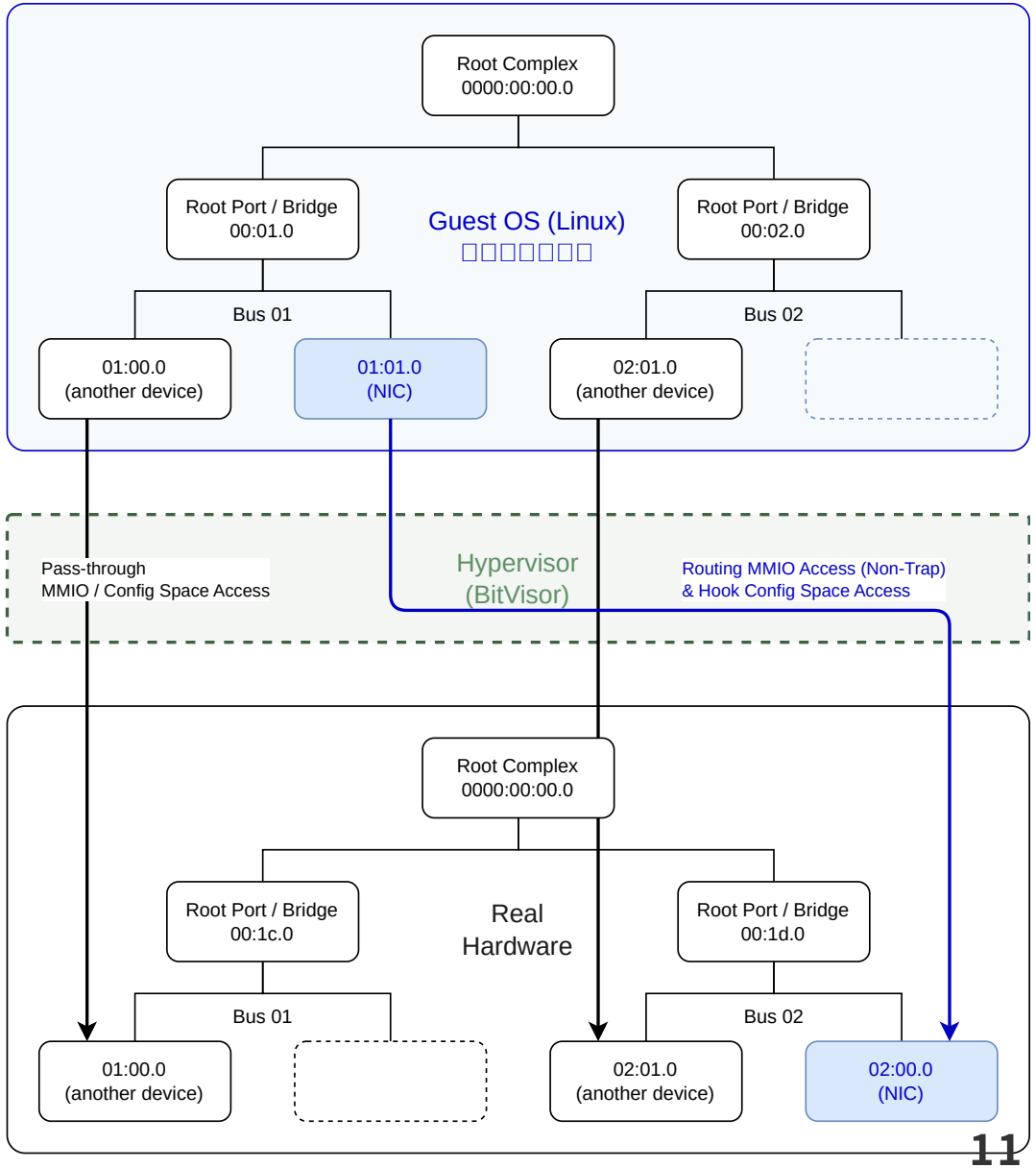
まとめ

PCIデバイスのBDFを誤魔化すハイパーバイザ

- BDFを誤魔化すだけでなく,
  Configuration空間, 特にBARをいい感じに
  エミュレートし, MMIOアクセスをねじ曲げる
  必要があった
- 電源管理などの問題もあったが,
  強硬手段で解決

ポイント

- PCI Bridgeの仮想化などの大規模な
  エミュレーションはせず, 基本的にはホストの
  構成をそのまま利用する
- 最小限のフックで, ほとんどパススルーで動く



おっと, 時間が余っていますね...?

ここからは... 趣味の話をしよう

【令和最新版】 ARM開発環境 Apple Silicon Mac

お手軽開発ボードM1 MacBookで自作OSを動かそう！

petitstrawberry

2026/05/30 Kernel/VM探検隊@関西 12回目

自作OSを実機で動かしたい

みなさん、自作OSの開発をやっていると思うのですが、開発環境、実行環境はどうしていますか？

- QEMUなどのエミュレータで動かす
- 実機で動かす
  - JTAGなど利用可能な開発ボードで動かす
  - printデバッグで頑張る

こんな感じでしょうか？

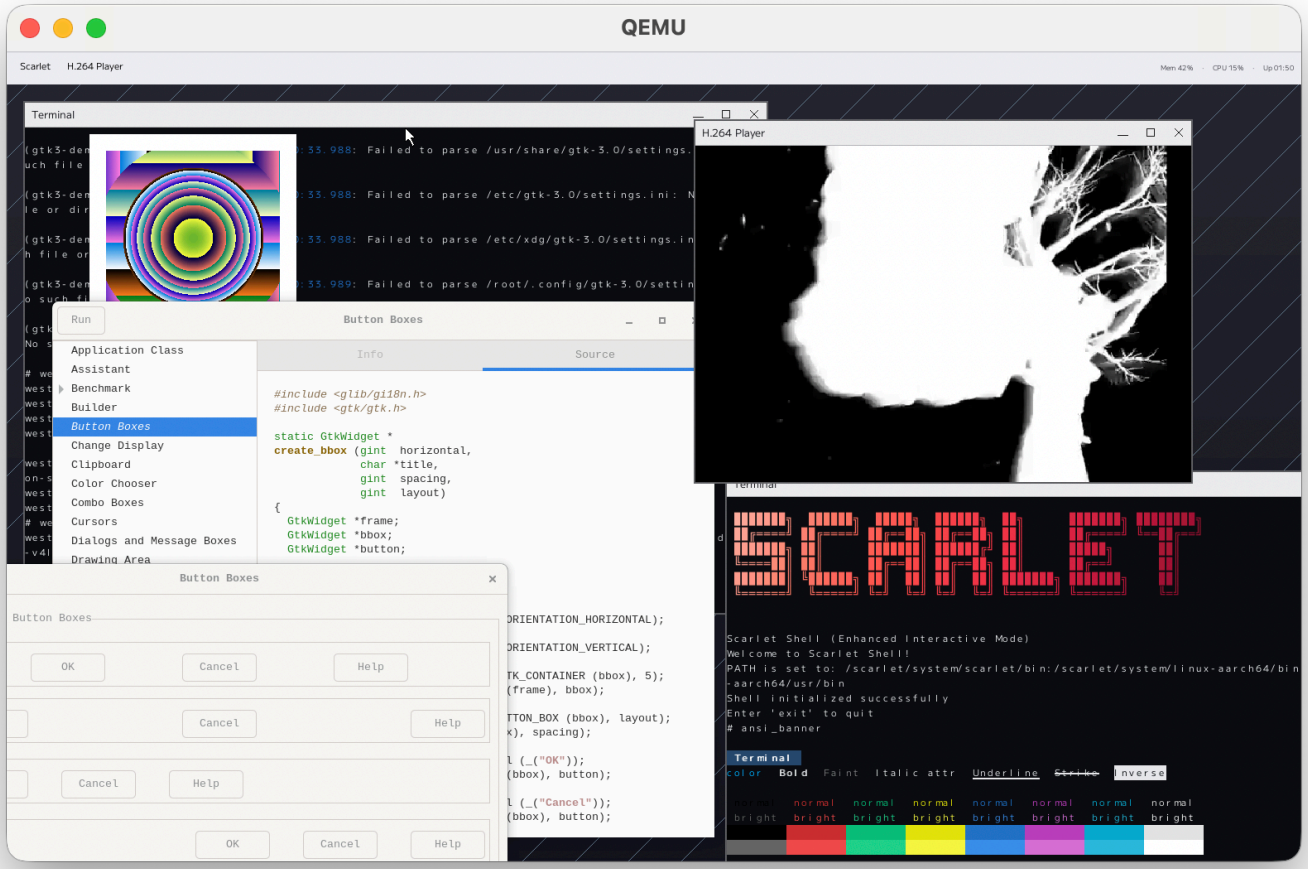
私は, これまで主にQEMUで動かして開発していましたが, いよいよ実機で動かしたくなってきました

今回は, その時に選んだ環境と, そこでの開発の様子を紹介します

ちなみに... 動かしたい自作OS Scarlet[2]

思いついたことを詰め込んだキメラのRust製OS

- **複数OS向けバイナリの透過的な実行**
  - ユーザやプログラムは何も意識せずに
    execやshellを使うだけで,
    色々なOS向けのバイナリが動く
    - Scarlet
    - Linux (GUI含む)
    - xv6
    - Windows, macOS (WIP)
- **ユーザランド含めて一通りの機能**
  - GUI, Video, Audio, Network, etc
- **謎のハイパーバイザ機能**
  - 何気にType-2ハイパーバイザでもある
  - KVM APIも再現してるので,
    先述の互換機能と合わせて
    Linux KVM前提のユーザランドVMMが動く
    (Firecrackerとか)



GitHub:
petitstrawberry/Scarlet

[2] Scarlet: https://github.com/petitstrawberry/Scarlet

良さげな開発ボードを求めて

いい感じの実機で動かしたい

条件

- ARM 64bit
  - 自作OSの対応アーキテクチャがRISC-VとARMなので (RISC-Vはそこそこ環境を持っている)
- UARTとデバッガが使える
  - USBケーブル一本で両方できると嬉しい (**令和なので**)
- そこそこ実用的な性能
  - なんなら, 高性能がいい
- 搭載デバイスや, ブート方法などのドキュメントが充実していると嬉しい
- UEFIでブートできると嬉しい
  - ボード依存のブートローダを作るのは面倒くさいので, できればUEFIで, さらにLimineなどの汎用的なブートローダも動くと嬉しい

そんな, 都合の良い開発ボードがあるのか...?

決定

最高のARM開発ボードは, Apple Silicon Macである(決定)

ここがすごい！

- 最高の性能
  - **開発ボード**とは思えない,
    実用的で今でも最前線で戦える**圧倒的性能**
- ラップトップもあるという珍しいタイプの
  ロマンのある**開発ボード**
  - ミニPCや高解像度モニター付きデスクトップタイプも！
- **なぜか**世界でバカ売れしているため, 価格もこなれているし, 在庫も豊富
- デバイスの仕様が公開され, ドキュメントもあるし, 実際にドライバの実装も公開中！？
- UEFIでブートできる
- USB-Cケーブル一本でUARTも取れるし, ペイロード転送やデバッグもできる

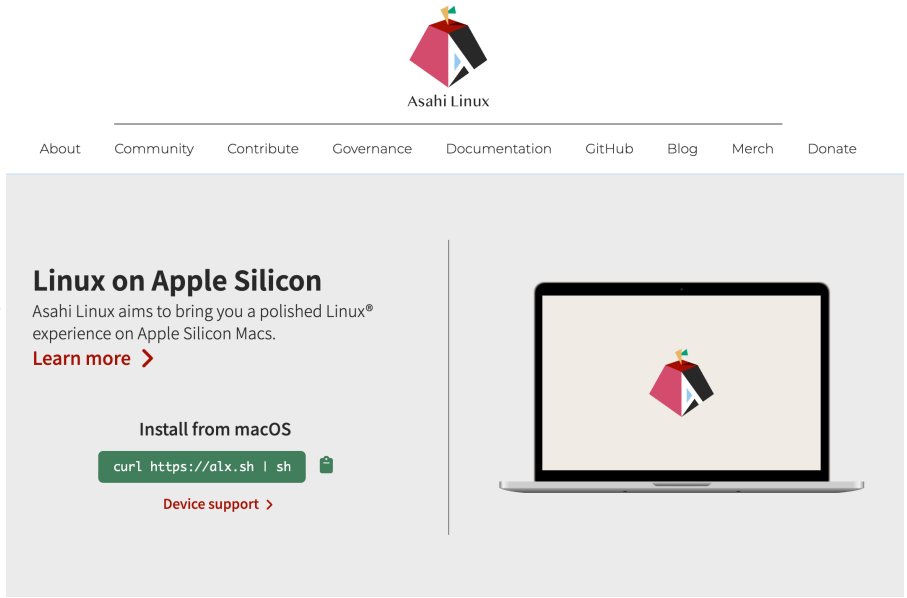
ありがとうApple...???

ありがとう Asahi Linux

という茶番は置いておき, 実際にこの便利環境を実現しているのは, Apple Silicon MacにLinuxを移植している**Asahi Linuxプロジェクトのおかげである**

Asahi Linuxとは

「Asahi Linuxは、2020年モデルのM1搭載Mac mini, MacBook Air, MacBook Proを皮切りに、Apple Silicon搭載MacにLinuxを移植することを目標とするプロジェクトおよびコミュニティです。」[3]



- 実装だけでなく, リバースエンジニアリングの結果などもドキュメントとして公開されてる (ブログもある)
- ブートフローも明確 / デバイスの仕様もわかる
- ファンのおかげで逆にそこら辺の開発ボードより充実した環境

[3]: Asahi Linux, https://asahilinux.org/

その中で, ブートローダである**m1n1**[4]が開発された

Apple M1 Boot Flow

```
+--------+    +--------+    +--------+    +--------+    +----------+
| iBoot  | -> | m1n1   | -> | U-Boot | -> | GRUB   | -> | Linux    |
+--------+    +--------+    +--------+    +--------+    +----------+
Apple         SoC init      Runs after    EFI app        Linux
boot          hand-off      m1n1          loader         Kernel
```

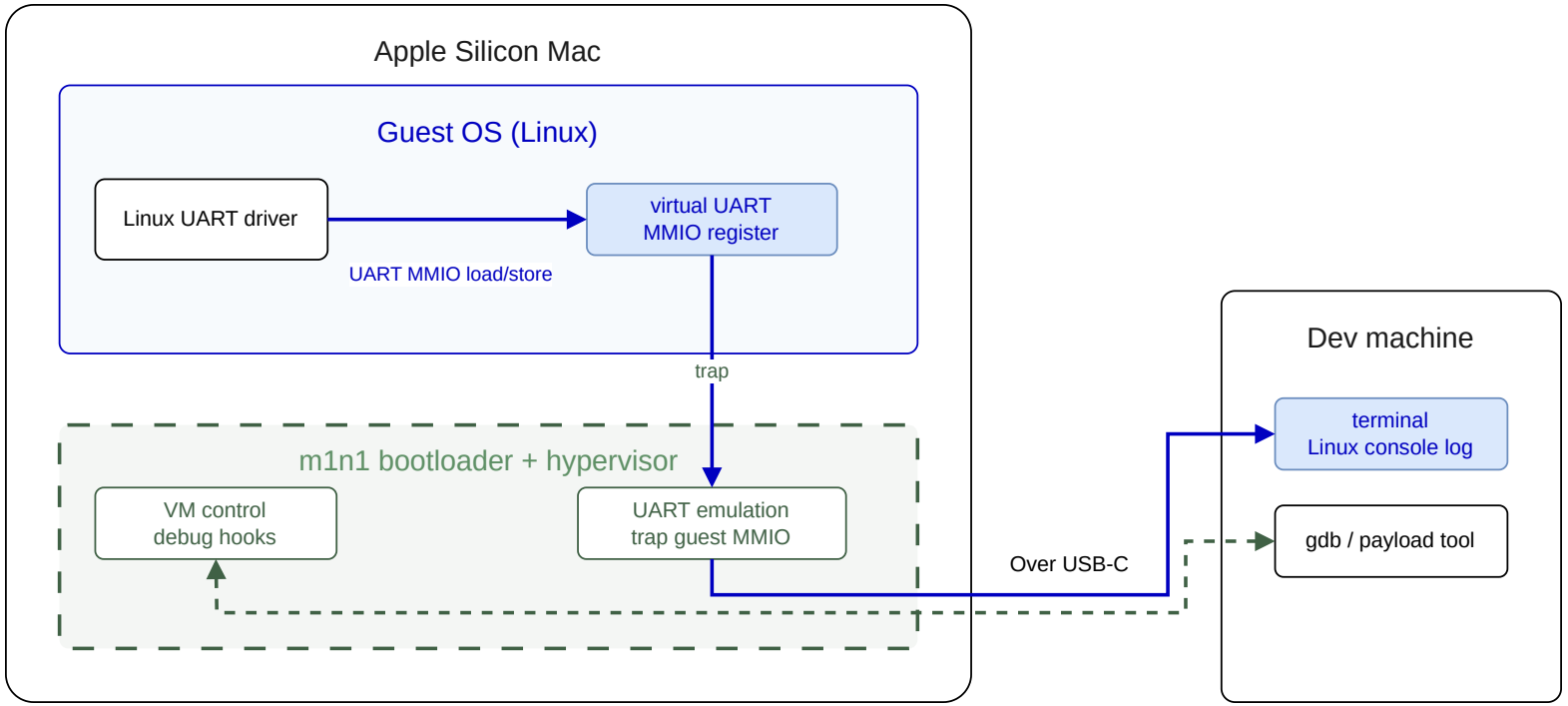
- ブートローダ**としての**機能
  - デバイスの初期化あたりをやってくれる
  - framebufferとかも用意してくれる
  - U-Bootなどを起動可能に
    - ここからUEFI互換でEFIアプリまで行ける (GRUBなどのブートローダが使える！)

しかし, このm1n1は, 単なるブートローダではない！

[4]: m1n1, https://github.com/AsahiLinux/m1n1

開発用ハイパーバイザとしてのm1n1

m1n1は, ブートローダであると同時に, **Apple Silicon Macのハイパーバイザ**でもある



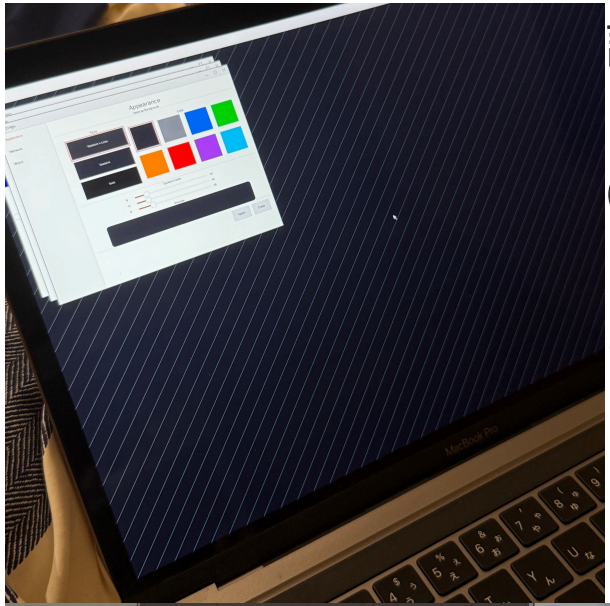
- ゲストOSのUARTへのアクセスをフックして, USB-C経由で開発マシンに転送する機能
- ペイロード転送機能
  - m1n1以降のU-Boot, カーネルイメージなどを転送して起動できる
- gdb stub機能
  - USB-C経由で, gdbやlldbから接続して, ブレークポイントを仕掛けたり, メモリを覗いたり

令和最新版 ARM開発フロー

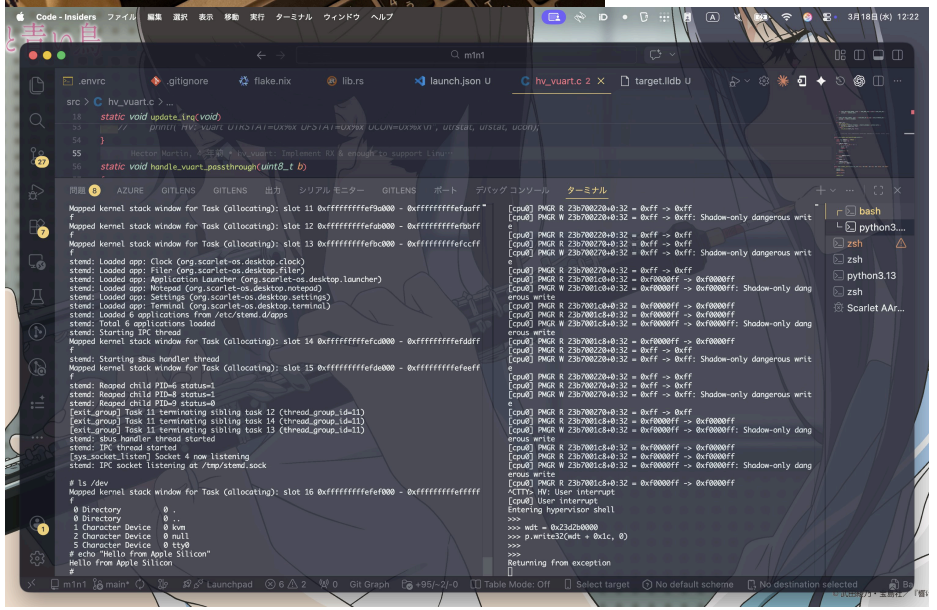
1. MacBookを再起動
2. m1n1が起動 (ここでUSB-C経由で開発マシンを接続する時間がある)
3. 開発マシンからU-Boot+GRUB/Limineなどのブートローダ+自作OSのイメージを転送
4. VMが起動してU-Bootへ --> GRUB/Limineへ --> 自作OSが起動
5. 開発マシンからUART経由でOSのログを見たり, gdbでデバッグしたりする

USBケーブル1本で全て完結...

これでええやん...!　最高



謎のAIC (Apple Interrupt Controller)や, IOMMU, UARTあたりのドライバを実装



おわりに

前編

- PCIデバイスのBDFを誤魔化すハイパーバイザを作った
  - BDFを誤魔化すだけでなく, BARも誤魔化す必要があった
  - 電源管理などの問題もあったが, 強硬手段で解決

後編

- 令和最新版ARM開発ボードApple Silicon Macを紹介
- Asahi Linuxプロジェクトとm1n1ブートローダを紹介