電子辞書で学ぶ Linux のサスペンド

末田卓巳 aka @puhitaku

情報科学若手の会 春の陣 2023 #wakate2023s

自己紹介

末田 卓巳 Takumi Sueda @puhitaku

仕事

- フリーランス
- 米住宅ベンチャー HOMMA Inc.
- 執筆
- セキュリティキャンプ講師
- OSS ライセンスコンサル etc.

好きなもの

- 全レイヤーの技術、特に低いもの
- リバースエンジニアリング
- 3Dプリント
- 音楽

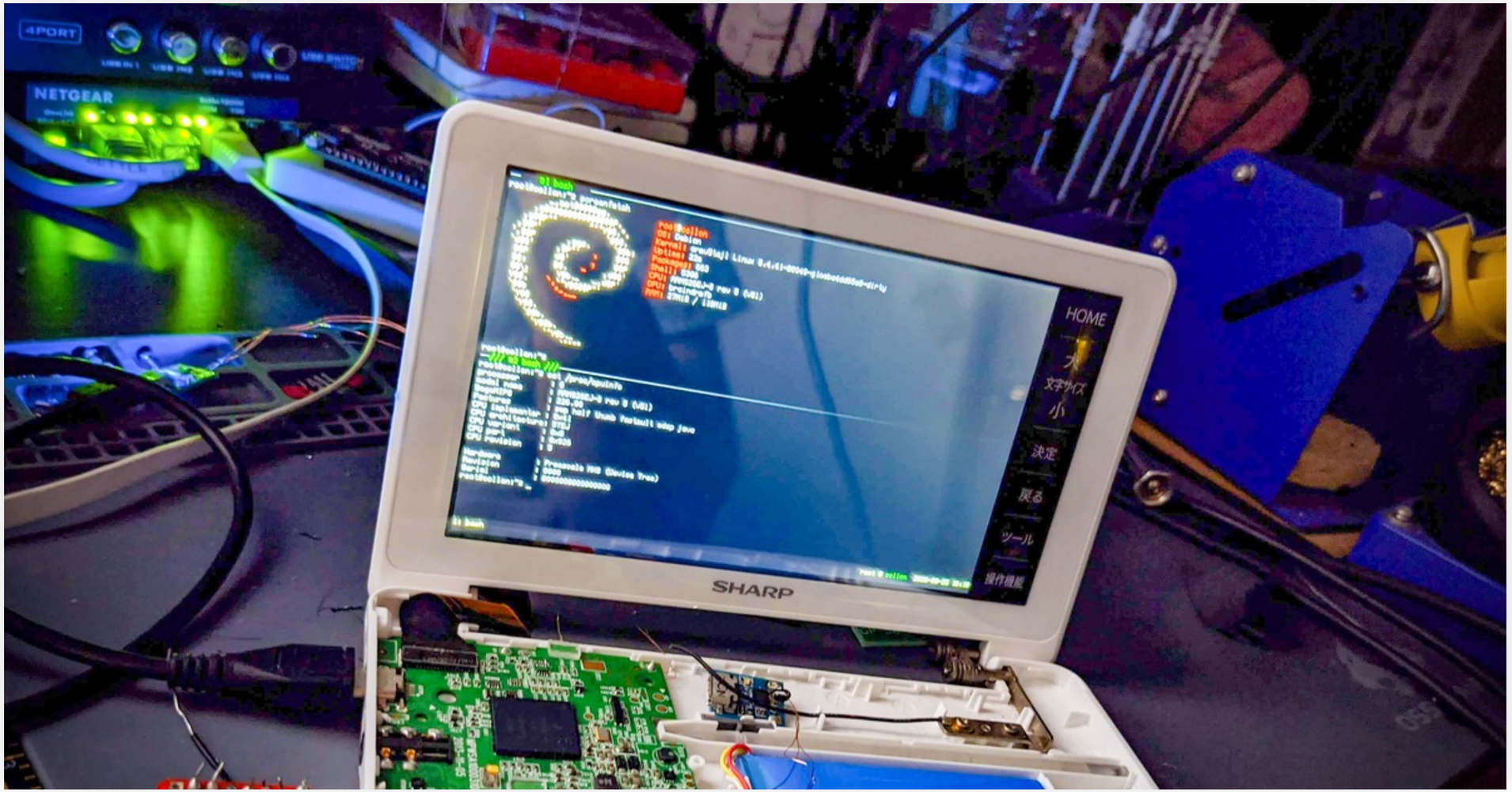
若手の会参加歴

- 2014年よりほぼ毎年参加

自己紹介

日経 Linux で連載「電子辞書で Linux を動かそう」を執筆中！第3回が来月発売の5月号に載ります！

SHARP Brain とは？



電子辞書 SHARP Brain

SHARP Brain とは？

特徴

- SHARP が2008年に発売した電子辞書ブランド
- Windows CE を搭載（2020年の機種まで）
- CE 向けアプリの一部が動く
- 自分でコンパイルした exe も動く
- 発売後すぐ Brain ハック界隈が 2ch で成立

Brain で動く Windows CE アプリ

- アプリランチャー
- オフラインで Wikipedia を読むソフト
- matplotlib
- ギャルゲー

などなど…

SHARP Brain PW-SH1

SHARP Brain とは？

CPU: NXP i.MX283
ARM926EJ-S, armv5tej
454 MHz

DRAM: LPDDR 128MB

LCD: 800x480 など

SD: SDXC
スロットに挿抜可能

eMMC: 8GB
SDの親戚・挿抜不可

その他:
バッテリー、タッチパネル、
磁気センサー（開閉検知）など

ハードウェアの構成としては 初代 Raspberry Pi をもっと古く素朴にして
キーボードと LCD とバッテリーを付けた感じ

SHARP Brain とは？

U-Boot と Linux のポーティング

- 2020年に Brain にブートローダ「U-Boot」と Linux カーネルを移植
- Raspberry Pi と同等の簡単さ（SD に焼くだけ）で使える Linux ディストリビューションを配布

実装したドライバ・実現している機能

- LCD（新規実装）
- キーボード（新規実装）
- タッチパネル
- SD / eMMC 読み書き
- 圧電素子によるビープ音

etc ...

SHARP Brain PW-SH1

SHARP Brain とは？

実現していない機能

- パワーマネジメント
  - スリープ
  - cpufreq（CPU 周波数制御）
- サウンド

SHARP Brain PW-SH1

SHARP Brain とは？

実現していない機能

- パワーマネジメント
  - スリープ
    - cpufreq（CPU 周波数制御）
- サウンド

SHARP Brain PW-SH1

「スリープ」とは？

「スリープ」とは？

Linux の「スリープ」にはいくつか種類がある

- Suspend-to-Idle
- Standby
- Suspend-to-RAM
- Hibernation

上記を総称して "sleep", "system-wide sleep" 等と呼び、
一般に、下に行くほど低消費電力

「スリープ」とは？

1. Suspend-to-Idle

CPU を暇にするだけの
純ソフトウェア的なスリープ

😴 Userspace（プロセスやスレッド）
😴 Timekeeping（tick, コンテキストスイッチ）
😴 I/O

2. Standby

ブートに使われていない CPU を寝かし
もっと電力を削る

😴 Suspend-to-Idle でオフになるもの全部
😴 Non-boot CPU

3. Suspend-to-RAM

CPU とデバイスの状態を DRAM に置き
DRAM と復帰に必要なハード以外を寝かす

😴 Standby でオフになるもの全部
😴 ほぼすべてのハードウェア

4. Hibernation

CPU とデバイスの状態を
永続化可能な記録媒体に逃がしシャットダウン

😴 すべてのハードウェア

「スリープ」とは？

1. Suspend-to-Idle

CPU を暇にするだけの
純ソフトウェア的なスリープ

😴 Userspace（プロセスやスレッド）
😴 Timekeeping（コンテキストスイッチ）
😴 I/O

2. Standby

ブートに使われていない CPU を寝かし
もっと電力を削る

😴 Suspend-to-Idle でオフになるもの全部
😴 Non-boot CPU

3. Suspend-to-RAM

CPU とデバイスの状態を DRAM に置き
DRAM と復帰に必要なハード以外を寝かす

😴 Standby でオフになるもの全部
😴 ほぼすべてのハードウェア

電力削減量や復帰にかかる時間の
バランスを考えると電子辞書では
Suspend-to-RAM をやりたい！

Suspend-to-RAM 移行と復帰の流れ

Suspend-to-RAM へ移行する流れ

1. システム全体にサスペンド通知を送り、カーネルの各サブシステムをスリープに備えさせる
2. タスクをフリーズする
3. デバイスをサスペンド & 復帰に使うデバイス以外の割り込みをハンドルしないよう設定
4. Non-boot CPU を止める（それ以外の CPU のタスクと IRQ は Boot CPU にマイグレートされる）
5. スケジューラーの tick を無効化しコンテキストスイッチ等を止める
6. プラットフォーム固有のファームウェア等に制御を渡して RAM 以外は低消費電力状態へ移行もしくは電源断
7. 復帰に使うデバイス（キーボードなど）からの割り込みが来るまで寝る

Suspend-to-RAM から復帰する流れ

1. 復帰に使うスイッチやデバイス（キーボードなど）からの割り込みが来る
2. プラットフォームによって異なる過程を経て CPU が復帰し割り込みがハンドルされる
   - 復帰しなくていい割り込みであれば再度寝る
3. カーネルに制御が戻る
4. カーネルのコア、tick、スケジューリングを復帰させる
5. Non-boot CPU を起こす
6. デバイスを起こし IRQ をもとに戻す
7. タスクを復帰 (thaw =「雪解け」) させる
8. システム全体に復帰通知を送る

新たなハードで Suspend-to-RAM をやるには何が必要か？

Suspend-to-RAM をやるには何が必要か？

Suspend-to-RAM に必要な2つの要素

パワーマネジメント

サスペンド通知が来たら graceful に電源を切る関数をドライバの dev_pm_ops 構造体に実装

デバイスとレギュレータ（電源）の関係を Device Tree で記述

たとえば、Brain なら…

- SoC 内蔵レギュレータ
- 外部ハードの電源を制御する MOSFET のゲート端子に接続された GPIO
- SPI, I^2C 等のペリフェラル

復帰に使うデバイスの割り込みハンドラ

復帰すべき入力が来た時にデバイスドライバから Power Management Subsystem へ復帰を指示する手続きを ISR に実装

たとえば、Brain なら…

- 画面開閉を検知する磁気センサー
- 電源ボタンが繋がる GPIO

Suspend-to-RAM をやるには何が必要か？

Suspend-to-RAM を実行する側のコードと Device Tree

- state_store (/kernel/power/main.c)
  - pm_suspend (/kernel/power/suspend.c)
    ⋮
    - suspend_devices_and_enter
      - platform_suspend_begin
      - suspend_console
      - suspend_enter
        - suspend_ops->enter
          → ここでサスペンドへ

/sys/power/state への write を起点とする
コールスタック概観

```
panel {
        compatible = "sii,43wvf1g";
        backlight = <&backlight_display>;
        dvdd-supply = <&reg_lcd_3v3>;
        avdd-supply = <&reg_lcd_5v>;
```

i.MX28 開発ボードの Device Tree のうち
LCD と電源の関係を記した箇所

この情報をカーネルが見て電源を入切する

Suspend-to-RAM をやるには何が必要か？

復帰に使うデバイスの割り込みハンドラと
Suspend-to-RAM を実行するコード間の気持ちの伝達

1. ドライバの割り込みハンドラが
pm_wakeup_event 関数を呼び
PM subsystem 内部のカウンタを
インクリメントする

```
if (!bdata->key_pressed) {
        if (bdata->button->wakeup)
                pm_wakeup_event(bdata->input->dev.parent, 0);
```

/drivers/input/keyboard/gpio_keys.c L443 (Linux 5.4.234)

2. suspend_enter は CPU の PC が
戻ってくるたびにカウンタを
確認し、増えていればループ脱出
増えてなければ再度 CPU を寝かす

```
do {
        error = suspend_enter(state, &wakeup);
} while (!error && !wakeup && platform_suspend_again(state));
```

/kernel/power/suspend.c L502 (Linux 5.4.234)

つまり復帰に関わらない割込は無視

Brain における実装状況

Brain における実装状況

ひとまず Suspend-to-RAM はできたが道半ば

- ✅ Suspend-to-RAM の実行
- ✅ 電源ボタンによる復帰（キーマトリクスを GPIO 直接続で読む機種）
- ⬜ 電源ボタンによる復帰（キーイベントを MCU から I^2Cで読む機種）
  - I^2C ペリフェラル単位の割り込みになっててどのキーを押しても起きてしまう
  - ほとんどのペリフェラルが寝ている中 ISR でできることは極めて限定的
  - Windows だと実現しているのでリバースエンジニアリングが必要？
- ⬜ LCD 等の電源制御
  - FET のゲートや ENABLE 端子につながる GPIO を regulator として Device Tree に表記？
- ⬜ 省電力性の検証
- ⬜ OS イメージとしてリリース

質疑応答ログ

Q. kthread と userspace のプロセスはどういう順番で凍る？

suspend_prepare → suspend_freeze_processes [1]曰く userspace を先に凍らせる。
[1] https://elixir.bootlin.com/linux/v5.4.234/source/kernel/power/power.h#L250

Q. 4つの手法によって消費電力低減効果はどれくらい差がある？

システムによる。たとえばめちゃくちゃでかいウエハの SoC と少ない周辺ハードの組み合わせであれば Suspend-to-Idle だけでもインパクトが大きいし、Brain のような小さい SoC ならシステム全体の消費電力で 115mA → 86mA 程度しか差が出ない[1]。シングルコアの Brain にとって Standby は Suspend-to-Idle と差がない。Suspend-to-RAM はこれから LCD の電源などインパクトの大きい部分の実装をやるのでそれ以降計測し判明予定。
[1] https://twitter.com/puhitaku/status/1629849862924648455

出典・参考

- Bootlin Elixir: https://elixir.bootlin.com/linux/v5.4.234/
- Mainline Linux
  - /Documentation/admin-guide/pm/suspend-flows.rst
  - /Documentation/admin-guide/pm/sleep-states.rst