目次

NuttX のファームウェア開発

meets

MA-S1xx シリーズは Linux のほかに、OS として NuttX を利用したファームウェアで運用することもできます。

NuttX を採用したファームウェアを使用すると、下記のようなメリットがあります。

- 高速起動
  - 2秒でアプリケーションが起動
  - PPP 接続をする場合、LTE module 起動まで 18 秒ほど、PPPは 20 秒ほどで接続完了
    - ⇒ 動作している時間を Linux と比べて非常に短くできるため、バッテリー駆動の時間を大幅に延ばすことが可能
- 高速シャットダウン(Filesystem を unmount するだけ)
- ちいさいファームウェアサイズ (0.5 MiB ～ 2MiB 程度)

実際のファームウェアサイズの例はこのようなサイズ感です。

[参考]

```
nuttx$ ls -l uImage
-rw-rw-r-- 1 user user 1706160  2月 29 07:08 uImage
```

このファームウェアでは、下記のような処理を行っています。

- RTC Alarm で定期的に起動して、下記処理を行う
  - RS485 接続のセンサーに電源を供給
  - センサーの起動を待ち、独自プロトコルのセンサーからデータを取得
  - eMMC にデータを保存 (1)
- 送信するタイミングの場合、下記を追加で行う
  - LTE module の電源を投入
  - LTE module が起動したら、PPP 接続の設定を行い、LTE 網に Attach されるのを待つ
  - LTE 網に Attach されたら、PPP 接続を行う
  - PPP 接続完了後、(1) で保存されたデータを読み込み、JSON 形式にして SORACOM Funk へ送信する
  - HTTP Response Header にある日時情報を利用して、RTC へ時刻同期を行う
  - HTTP Response Body の情報を見て、RTC Alarm のインターバルを変更する
- 上記処理完了後、RTC Alarm の設定を行い、Filesystem を unmount 後、shutdown する

上記のメリットに加えて、Nim 言語 での開発も可能になっているので、

- Nim の各種モジュールを使用して簡単にアプリケーションを作成可能
- Python で開発するのと同じ程度の記述量で RTOS のアプリケーションを作成可能
- C 言語にコンパイルされるため、Python や microPython などと比較して超高速で処理が可能
- C 言語の関数をバインディングモジュールを作る必要がなく簡単に呼び出すことが可能
    - Nim のモジュールにない、NuttX 独自の関数を簡単に使用することができる

となっています。

Nim での記述例

RTOS 向けとは思えないほどの少ないコードで記述できます。

SORACOM Harvest に bool 値の配列を JSON で送信する

```
import std/asyncdispatch
import std/httpclient
import std/json
import std/osproc
import std/strformat

proc post_harvest(data: seq[bool]) {.async.} =
  let client = newAsyncHttpClient()
  defer: client.close()
  client.headers = newHttpHeaders({"Content-Type": "application/json"})
  let body = %* {"data": data}
  echo $body
  try:
    let resp = await client.request("http://harvest.soracom.io",
        httpMethod = HttpPost, body = $body)
    echo &"result: {resp.code}"
    let resp_headers = resp.headers
    for key, val in resp_headers.pairs:
      echo &"key: {key} => value: {val}"
  except:
    let errmsg = getCurrentExceptionMsg()
    echo &"error: {errmsg}"
```

Modbus-RTU でデータを取得

```
import std/asyncdispatch
import std/strformat
import std/times
import nim_asyncmodbus

proc get_data(mb: ModbusRtu, address: uint8) {.async.} =
  echo "--- Input Status"
  let hregs = await mb.read_input_bits(address, 1.uint16, 8)
  echo &"--- [{address}]: Input Bits -> count: {hregs.len}"
  echo hregs
  try:
    await post_harvest(hregs)
  except:
    let errmsg = getCurrentExceptionMsg()
    echo &"post_harvest: {errmsg}"

proc mb_task() {.async.} =
  let mb = newModbusRtu("/dev/ttyFC1", 19200)
  echo "mb instanciated."
  discard mb.set_slave(2)
  echo "set_slave() -> completed"
  discard mb.connect()
  echo "connected"

  for i in 0 ..< 2:
    let time_start = now()
    await mb.get_data(2)
    let time_end = now()
    let elapsed = time_end - time_start
    echo &"elapsed: {elapsed}"
    await sleepAsync(100)
  mb.close()
```

開発方法

- NuttX on MA-S1xx