リストA Cargo.tomlにmutex-traitの依存を追加

```
toml
mutex-trait = "0.2"
```

リストB ヘッダ・ファイルFreeRTOS/semphr.h

```
#define xSemaphoreCreateMutex()    xQueueCreateMutex( queueQUEUE_TYPE_MUTEX )
#define xSemaphoreTake( xSemaphore, xBlockTime )    xQueueSemaphoreTake( ( xSemaphore ), ( xBlockTime ) )
#define xSemaphoreTakeFromISR( xSemaphore, pxHigherPriorityTaskWoken )
xQueueReceiveFromISR( ( QueueHandle_t ) ( xSemaphore ), NULL, ( pxHigherPriorityTaskWoken ) )
```

リストC embedded-halが2バージョンあるCargo.toml

```
[dependencies]
embedded-hal = "=1.0.0-alpha.9"
embedded-hal-0-2 = { package = "embedded-hal", version = "0.2.7", features = ["unproven"] }
esp-idf-sys = { version = "=0.32", features = ["binstart"] }
esp-idf-svc = { version="=0.45", features = ["experimental", "alloc"] }
# 以下略
```

リストD main関数

```
#[toml_cfg::toml_config]
pub struct Config {
    #[default("")]
    wifi_ssid: &'static str,
    #[default("")]
    wifi_psk: &'static str,
}

use esp_idf_hal::peripherals::Peripherals;
use esp_idf_hal::prelude::*;
use esp_idf_hal::{gpio::*, i2c, spi};

mod client;
mod macros;
mod mutex;
mod sensor_value;
mod server;
mod wifi;

fn main() -> anyhow::Result<()> {
    esp_idf_sys::link_patches();
    esp_idf_svc::log::EspLogger::initialize_default();

    let peripherals = Peripherals::take().expect("never fail");

    let _wifi = wifi::connect(peripherals.modem, CONFIG.wifi_ssid, CONFIG.wifi_psk)?;

    let spi = peripherals.spi2;
    let sclk = peripherals.pins.gpio3;
    let sdo = peripherals.pins.gpio4;
    let rst = PinDriver::output(peripherals.pins.gpio7)?;
    let dc = PinDriver::output(peripherals.pins.gpio8)?;
    let sdi_not_used: Option<Gpio0> = None;
    let cs_not_used: Option<Gpio0> = None;

    let config = spi::SpiConfig::new().baudrate(4.MHz().into());
```

```
let spi = spi::SpiDeviceDriver::new_single(
    spi,
    sclk,
    sdo,
    sdi_not_used,
    spi::Dma::Disabled,
    cs_not_used,
    &config,
)?;
server::spawn_server(spi, rst, dc)?;

let i2c = peripherals.i2c0;
let sda = peripherals.pins.gpio0;
let scl = peripherals.pins.gpio1;

let config = i2c::I2cConfig::new().baudrate(100.kHz().into());
let i2c = i2c::I2cDriver::new(i2c, sda, scl, &config)?;
client::spawn_client(i2c, peripherals.pins.gpio9)?;

loop {
    std::thread::sleep(std::time::Duration::from_secs(1))
}
```

リストE macros.rs

```
#[macro_export]
macro_rules! no_std_anyhow {
    ($e:expr) => {
        ($e).map_err(|e| anyhow!("{:?}, {}:{}", e, std::file!(), std::line!()))
    };
}
```

リストF wifi.rs

```
use std::{net::Ipv4Addr, time::Duration};

use anyhow::bail;
use esp_idf_hal::peripheral;
use esp_idf_svc::netif::{EspNetif, EspNetifWait};
use esp_idf_svc::{eventloop::*, wifi::*};

use embedded_svc::wifi::*;

use anyhow::Result;
use log::*;

/// Connect to a Wi-Fi access point.
/// Returns initialized Wi-Fi stack on success.
///
/// - ssid: Your Wi-Fi name
/// - pass: Your Wi-Fi password
///
/// Note that ESP32-C3 does not support the 5GHz band, please use a WiFi with active 2.4GHz band.
pub fn connect<'a>(
    modem: impl peripheral::Peripheral<P = esp_idf_hal::modem::Modem> + 'static,
    ssid: &'a str,
    pass: &'a str,
) -> Result<Box<EspWifi<'a>>> {
    // 1. Initialize the Wi-Fi stack
    let sysloop = EspSystemEventLoop::take()?;
    let mut wifi = Box::new(EspWifi::new(modem, sysloop.clone(), None)?);
```

```
info!("Wifi created, about to scan");

// 2. Scan our Wi-Fi access point
let ap_infos = wifi.scan()?;

// 3. If our access point found, configure the channel
let ours = ap_infos.into_iter().find(|a| a.ssid == ssid);
let channel = if let Some(ours) = ours {
    Some(ours.channel)
} else {
    None
};

// 4. Configure the Wi-Fi stack as a Wi-Fi station
wifi.set_configuration(&Configuration::Client(ClientConfiguration {
    ssid: ssid.into(),
    password: pass.into(),
    channel,
    ..Default::default()
}))?;

// 5. Wait until Wi-Fi stack is operating
wifi.start()?;

if !WifiWait::new(&sysloop)?
    .wait_with_timeout(Duration::from_secs(20), || wifi.is_started().unwrap())
{
    bail!("Wifi did not start");
}

wifi.connect()?;

if !EspNetifWait::new::<EspNetif>(wifi.sta_netif(), &sysloop)?.wait_with_timeout(
    Duration::from_secs(20),
    || {
        wifi.is_connected().unwrap()
            && wifi.sta_netif().get_ip_info().unwrap().ip != Ipv4Addr::new(0, 0, 0, 0)
    },
) {
    bail!("Wifi did not connect or did not receive a DHCP lease");
}

info!("Wifi connected");

    Ok(wifi)
}
```

リストG sensor_value.rs

```
use serde::{Serialize, Deserialize};

/// Sensor value used between HTTP client and server communication.
#[derive(Serialize, Deserialize, PartialEq, Debug, Clone, Copy)]
pub(crate) enum SensorValue {
    Temperature(f32),
    Humidity(f32),
    Pressure(f32),
}

/// Implement ToString to draw a text in a display.
impl ToString for SensorValue {
    fn to_string(&self) -> String {
```

```
        match *self {
            Self::Temperature(t) => {
                format!("temp: {:.2}", t)
            }
            Self::Humidity(h) => {
                format!("humi: {:.2}", h)
            }
            Self::Pressure(p) => {
                format!("pres: {:.2}", p)
            }
        }
    }
}
```

リストH　HTTPサーバの実装server.rs

```
use embedded_graphics::{
    mono_font::{ascii::FONT_6X10, MonoTextStyle},
    pixelcolor::Rgb565,
    prelude::*,
    text::Text,
};
use embedded_hal_0_2::{blocking::spi, digital::v2::OutputPin};
use esp_idf_hal::delay::FreeRtos;
use esp_idf_sys::EspError;
use ssd1331::{DisplayRotation, Ssd1331};

use log::*;
use std::io::Read;
use std::marker::Send;
use std::sync::mpsc;
use std::thread;

use crate::no_std_anyhow;
use crate::sensor_value::*;
use anyhow::anyhow;
use embedded_svc::{http::server::*, io::Write};
use esp_idf_svc::http::server::*;

pub fn spawn_server<SPI, RST, DC>(spi: SPI, mut rst: RST, dc: DC) -> anyhow::Result<()>
where
    SPI: 'static + spi::Write<u8> + Send,
    <SPI as spi::Write<u8>>::Error: std::fmt::Debug,
    DC: 'static + OutputPin<Error = EspError> + Send,
    RST: 'static + OutputPin<Error = EspError> + Send,
{
    let (tx, rx) = mpsc::channel();

    let server_config = Configuration::default();
    let mut server = Box::new(EspHttpServer::new(&server_config)?);
    server
        .fn_handler("/", Method::Get, |req| {
            let html = index_html();
            req.into_ok_response()?.write_all(&html.as_bytes())?;

            Ok(())
        })?
        .fn_handler("/sensor", Method::Post, move |req| {
            use embedded_svc::io::adapters::ToStd;
            let mut body = String::new();
            // req.read_to_string(&mut body)?;
            ToStd::new(req).read_to_string(&mut body)?;
```

```
        let v: SensorValue = serde_json::from_str(&body)?;

        info!("posted value: {:?}", v);
        tx.send(v)?;

        Ok(())
    })?;
Box::leak(server);

thread::Builder::new()
    .stack_size(24000)
    .spawn(move || -> anyhow::Result<()> {
        let mut delay = FreeRtos;
        let mut disp = Ssd1331::new(spi, dc, DisplayRotation::Rotate0);
        no_std_anyhow!(disp.reset(&mut rst, &mut delay))?;
        no_std_anyhow!(disp.init())?;

        disp.clear();
        no_std_anyhow!(disp.flush())?;

        loop {
            let v = rx.recv()?;
            disp.clear();
            no_std_anyhow!(disp.flush())?;
            let text = v.to_string();
            let style = MonoTextStyle::new(&FONT_6X10, Rgb565::WHITE);
            no_std_anyhow!(Text::new(&text, Point::new(20, 30), style).draw(&mut disp))?;
            no_std_anyhow!(disp.flush())?;
        }
    })?;

    Ok(())
}

fn templated(content: impl AsRef<str>) -> String {
    format!(
        r#"
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>esp-rs web server</title>
    </head>
    <body>
        {}
    </body>
</html>
"#,
        content.as_ref()
    )
}

fn index_html() -> String {
    templated("Please post sensor value to /sensor")
}
```

リストI spawn_server()

```
pub fn spawn_server<SPI, RST, DC>(spi: SPI, mut rst: RST, dc: DC) -> anyhow::Result<()>
where
    SPI: 'static + spi::Write<u8> + Send,
    <SPI as spi::Write<u8>>::Error: std::fmt::Debug,
    DC: 'static + OutputPin<Error = EspError> + Send,
```

```
    RST: 'static + OutputPin<Error = EspError> + Send,
```

リストJ　HTTPクライアントclient.rs

```
use bme280::i2c::BME280;
use embedded_hal::i2c::I2c;
use esp_idf_hal::{delay::FreeRtos, gpio::*};
use esp_idf_svc::timer::*;

use crate::mutex;
use crate::no_std_anyhow;
use crate::sensor_value::*;
use anyhow::anyhow;
use log::*;
use mutex_trait::Mutex;
use std::ops::Deref;
use std::{sync::mpsc, sync::Arc, thread};

/// Sensor type that BME280 can measure.
/// Used to select a type to be posted.
#[derive(PartialEq, Clone, Copy)]
pub(crate) enum SensorType {
    Temperature,
    Humidity,
    Pressure,
}

impl SensorType {
    pub fn next(&self) -> Self {
        match *self {
            Self::Temperature => Self::Humidity,
            Self::Humidity => Self::Pressure,
            Self::Pressure => Self::Temperature,
        }
    }
}

pub fn spawn_client<I2C>(i2c: I2C, button_pin: Gpio9) -> anyhow::Result<()>
where
    I2C: 'static + I2c + Send,
{
    let mut button = Box::new(PinDriver::input(button_pin)?);
    button.set_pull(Pull::Down)?;
    button.set_interrupt_type(InterruptType::PosEdge)?;

    let state = Arc::new(mutex::Mutex::new(SensorType::Temperature));
    let s1 = state.clone();

    unsafe {
        button.subscribe(move || {
            s1.deref().lock(|s| {
                *s = s.next();
            });
        })?;
    }
    Box::leak(button);

    // http
    thread::Builder::new()
        .stack_size(8192)
        .spawn(move || -> anyhow::Result<()> {
            let (tx, rx) = mpsc::channel();
            let mut delay = FreeRtos;
```

```
// initialize the BME280 using the primary I2C address 0x76
let mut bme280 = BME280::new_primary(i2c);
no_std_anyhow!(bme280.init(&mut delay))?;

let timer = EspTaskTimerService::new()?;
let timer = timer.timer(move || {
    let measurements = no_std_anyhow!(bme280.measure(&mut delay)).unwrap();
    let value = match state.deref().lock(|s| *s) {
        SensorType::Temperature => SensorValue::Temperature(measurements.temperature),
        SensorType::Humidity => SensorValue::Humidity(measurements.humidity),
        SensorType::Pressure => SensorValue::Pressure(measurements.pressure),
    };
    match tx.send(value) {
        Ok(_) => {}
        Err(e) => {
            error!("channel operation failed {}", e)
        }
    }
})?;

timer.every(std::time::Duration::from secs(1))?;

            loop {
                let v = rx.recv()?;
                info!("{:?}", v);

                let resp = attohttpc::post("http://localhost/sensor")
                    .json(&v)?
                    .send()?;
                info!("Status: {:?}", resp.status());
            }
        })?;

    Ok(())
}
```

リストK　cfg.toml

```
[http-client]
wifi_ssid = ""
wifi_psk = ""
http_server = "http://localhost/sensor"
```

図A　起動ログ

```
I (6619) esp_idf_svc::http::server: Started Httpd server with config Configuration { http_port: 80,
https_port: 443, max_sessions: 16, session_timeout: 1200s, stack_size: 6144, max_open_sockets: 4,
max_uri_handlers: 32, max_resp_handlers: 8, lru_purge_enable: true, session_cookie_name: "SESSIONID" }
I (6649) esp_idf_svc::http::server: Registered Httpd server handler Get for URI "/"
I (6649) esp_idf_svc::http::server: Registered Httpd server handler Post for URI "/sensor"
```

図B　センサのログ

```
I (7769) http_app::client: Temperature(21.501692)
I (7779) esp_idf_svc::http::server: About to handle query string ""
I (7779) http_app::server: posted value: Temperature(21.501692)
I (8019) http_app::client: Status: 200
```

リストL　Cargo.toml

```
[http-client]
wifi_ssid = ""
wifi_psk = ""
http_server = "http:/<target ip>/sensor"

[http-server]
wifi_ssid = ""
wifi_psk = ""
```

図C　複数のシリアル・ポートが検出されるので，どちらを使うかコンソールで選ぶ

```
$ cargo espflash --monitor
Detected 2 serial ports. Ports which match a known common dev board are highlighted.
❯ /dev/ttyACM0 - USB_JTAG_serial_debug_unit
  /dev/ttyACM1 - USB_JTAG_serial_debug_unit
```

図D　HTTPサーバ側の書き込み

```
# server ディレクトリ
$ cargo espflash --monitor
# 中略
I (7521) esp_idf_svc::netif::status: Got IP event: DhcpIpAssigned(DhcpIpAssignment { netif_handle:
0x3fca3d08, ip_settings: IpInfo { ip: 192.168.100.24, subnet: Subnet { gateway: 192.168.100.1, mask:
Mask(24) }, dns: None, secondary_dns: None }, ip_changed: true })
```

リストM　cfg.toml

```
[http-client]
wifi_ssid = ""
wifi_psk = ""
http_server = "http:/<target ip>/sensor"
```

図E　HTTPクライアント側の書き込み

```
# client ディレクトリ
$ cargo espflash --monitor
```

リストN　CMakeLists.txt

```
idf_component_register(SRC_DIRS "src"
                       INCLUDE_DIRS "include"
                       )
```

リストO　include/my_component.h

```
/**
 * @brief print hello from C world!
 *
 */
void hello_from_c(void);
```

リストP　src/my_component.c

```
#include <my_component.h>
#include <esp_log.h>
```

```
static const char *TAG = "my_component";

void hello_from_c(void)
{
    ESP_LOGI(TAG, "hello!");
}
```

リストQ　src/main.rs

```
fn main() {
    esp_idf_sys::link_patches();
    unsafe { esp_idf_sys::my_component::hello_from_c() }
}
```

リストR　Cargo.toml

```
[[package.metadata.esp-idf-sys.extra_components]]
component_dirs = ["my_component"]
bindings_header = "my_component/include/my_component.h"
bindings_module = "my_component"
```