Tarefa Prática IOT 3

Nome: Carlos Delfino

Matricula: 202510110980368

Embarcatech Residência Profissional

Ano: 2025

Sumário

Sumário

Introdução

Foram desenvolvidos dois firmwares, um para enviar a situação do botão para o MQTT, e outro para receber o estado do botão, mudando o estado de um LED como indicativo.

Optei por usar meu próprio Broker, que foi implementado em mqtt://mqtt.rapport.tec.br utilizando o Musquitto, criei uma conta especial para uso do Monitor Wellingson:

Usário: wellingson

Senha: embarcawelingson492

O código está disponível no repositório git:

git@github.com:carlosdelfino/embarcatech_etapa_2_cap_2_Tarefa-Pr-tica-IOT-3.git

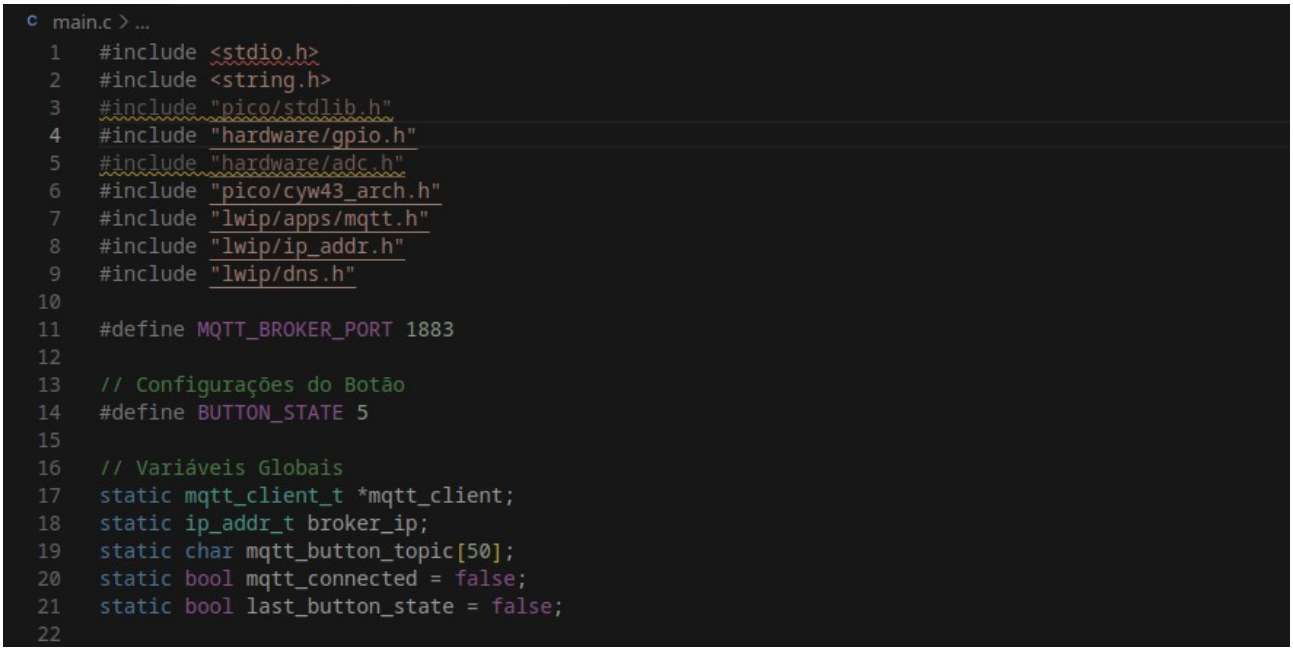
Abaixo apresento o código explicado.

Publisher (Servidor)

O Publisher é o código responsável por gerar os dados de telemetria do equipamento, através dele é obtido os dados no hardware e enviados separadamente para cada tópico no MQTT. Um publisher também pode ser um subscribe reagindo as atualizações no brocker envidas por outro publisher.

Carga das bibliotecas e frameworks e constantes

Na figura abaixo vemos as bibliotecas que foram importadas, e a declaração de cada variável necessária para o funcionamento da lógica. E duas macros (constantes) que identificam a porta do servidor MQTT e o pino do botão a ser usado.



Definição das assinaturas das funções

Logo a seguir temos as assinaturas das funções necessárias para organizar o código, trazendo assim clareza, declarar os callbacks.

Inicialização do WiFi, GPIO e cliente MQTT

Já na função **main()**, temos a inicialização do framework para uso do WiFi e do GPIO conforme o necessário para uso do botão. Também temos a inicialização do MQTT.

```c
int main()
{
    stdio_init_all();
    sleep_ms(ms: 2000);
    printf("\n=== Iniciando MQTT Button Monitor ===\n");

    // Inicializa Wi-Fi
    if (cyw43_arch_init()) {
        printf("Erro na inicialização do Wi-Fi\n");
        return -1;
    }
    cyw43_arch_enable_sta_mode();

    printf("[Wi-Fi] Conectando...\n");
    if (cyw43_arch_wifi_connect_timeout_ms(ssid: WIFI_SSID, pw: WIFI_PASSWORD, auth: CYW43_AUTH_WPA2
        printf("[Wi-Fi] Falha na conexão Wi-Fi\n");
        return -1;
    } else {
        printf("[Wi-Fi] Conectado com sucesso!\n");
    }

    // Configura GPIO do botão
    gpio_init(gpio: BUTTON_STATE);
    gpio_set_dir(gpio: BUTTON_STATE, out: GPIO_IN);
    gpio_pull_up(gpio: BUTTON_STATE); // <<< ATENÇÃO: pull-up ativado

    // Inicializa cliente MQTT
    mqtt_client = mqtt_client_new();
```

Resolução do nome do dominio do MQTT para IP

No código a seguir temos a resolução do nome do brocker mqtt, nela é cadastrado a função de callback **dns_check_callback** que trata a resolução do nome, caso tudo ocorra bem continua a execução.

```c
    // Resolve DNS do broker MQTT
    err_t err = dns_gethostbyname(hostname: MQTT_BROKER, addr: &broker_ip, found: dns_check_callback
    if (err == ERR_OK) {
        dns_check_callback(name: MQTT_BROKER, ipaddr: &broker_ip, callback_arg: NULL);
    } else if (err == ERR_INPROGRESS) {
        printf("[DNS] Resolvendo...\n");
    } else {
        printf("[DNS] Erro ao resolver DNS: %d\n", err);
        return -1;
    }
```

O loop principal (Super Loop)

Então entramos no loop principal (ou super loop) nele processamos o as tarefas que são periódicas, primeiro mantemos a conexão wifi sempre ativa, em seguida, obtemos o estado do botão e verificamos se foi alterado em relação a ultima interação, caso tenha sido alterado é chamada a função que interagem com o broker MQTT. Então aguardamos 1 segundo para a próxima interação.

```
    // Loop principal
    while (true) {
        // Atualiza tarefas de rede
        cyw43_arch_poll();

        // Lê o estado do botão
        bool button_state = !gpio_get(gpio: BUTTON_STATE); // <<< Inverte porque é pull-up

        // Se mudou de estado, publica
        if (button_state != last_button_state) {
            printf("[BOTÃO] Estado mudou para: %s\n", button_state ? "ON" : "OFF");
            publish_button_state(pressed: button_state);
            last_button_state = button_state;
        }

        sleep_ms(ms: 1000); // Ajuste conforme desejado
    }

    return 0;
}

```

Função responsável pela publicação do estado do botão

Na função **publish_button_state(bool)** é recebido um parâmetro que representa o estado do botão, ela verifica se o firmware está conectado ao broker MQTT, constrói o tópico MQTT que irá armazenar o estado do botão, e faz a publicação do tópico reservando o retorno da função para analisar se a publicação foi correta.

```
void publish_button_state(bool pressed) {
    if (!mqtt_connected) {
        printf("[MQTT] Não conectado, não publicando estado da porta\n");
        return;
    }
    char topic_button_state[50];
    snprintf(topic_button_state, sizeof(topic_button_state), "%s/button", mqtt_button_topic);

    const char *message = pressed ? "ON" : "OFF";

    printf("[MQTT] Publicando: tópico='%s', mensagem='%s'\n", topic_button_state, message);

    err_t err = mqtt_publish(client: mqtt_client, topic: topic_button_state, payload: message, payl

    if (err == ERR_OK) {
        printf("[MQTT] Publicação enviada com sucesso\n");
    } else {
        printf("[MQTT] Erro ao publicar: %d\n", err);
    }
}
```

Tratando a resolução de nome de domínio para IP

A função **dns_check_callback(const char, const ip_addr_t, void)** que é responsável pelo tratamento da obtenção do número IP do servidor.

```c
// Callback de DNS
void dns_check_callback(const char *name, const ip_addr_t *ipaddr, void *callback_arg) {
    if (ipaddr != NULL) {
        broker_ip = *ipaddr;
        printf("[DNS] Resolvido: %s -> %s\n", name, ipaddr_ntoa(addr: ipaddr));

        struct mqtt_connect_client_info_t ci = {
            .client_id = "pico-client",
            .keep_alive = 60,
            .client_user = NULL,
            .client_pass = NULL,
            .will_topic = NULL,
            .will_msg = NULL,
            .will_qos = 0,
            .will_retain = 0
        };

        printf("[MQTT] Conectando ao broker...\n");
        mqtt_client_connect(client: mqtt_client, ipaddr: &broker_ip, port: MQTT_BROKER_PORT, cb: mqtt
    } else {
        printf("[DNS] Falha ao resolver DNS para %s\n", name);
    }
}
```

Tratando a conexão com o broker MQTT

Já a função **mqtt_connection_callback(mqtt_client_t, void, mqtt_connection_status_t)** trata a conexão com o broker setando o flat mqtt_connected conforme o resultado da tentativa de conexão.

```c
// Callback de conexão MQTT
static void mqtt_connection_callback(mqtt_client_t *client, void *arg, mqtt_connection_status_t st
    if (status == MQTT_CONNECT_ACCEPTED) {
        printf("[MQTT] Conectado ao broker!\n");
        mqtt_connected = true;
    } else {
        printf("[MQTT] Falha na conexão MQTT. Código: %d\n", status);
        mqtt_connected = false;
    }
}
```

Subscriber (Cliente)

O Subscriber irá monitorar o broker para reagir as atualizações do tópico de interesse. E como dito no Publisher também pode se comportar como um Publisher.

Como estou aprendendo, cada código adoto uma abordagem nova no intuito de analisar sugestões e exemplos, tendo assim uma diversificação na lógica usada.

Estrutura que representa o estado do cliente

Já no subscriber adotei um struct para conter o estado do cliente e informações da conexão, como informações do cliente, os dados recebidos do tópico, o tópico que está sendo tratado naquele momento, seu comprimento, o endereço do broker se está conectado, quantos subscribers estão conectados, aqui neste parâmetro mostra que um subscriber também pode ser um publisher.

```c
typedef struct {
    mqtt_client_t* mqtt_client_inst;
    struct mqtt_connect_client_info_t mqtt_client_info;
    char data[MQTT_OUTPUT_RINGBUF_SIZE];
    char topic[MQTT_TOPIC_LEN];
    uint32_t len;
    ip_addr_t mqtt_broker_address;
    bool connect_done;
    int subscribe_count;
    bool stop_client;
} MQTT_CLIENT_DATA_T;
```

Definição de constantes relativas ao cliente (Subscriber)

Como pode ser visto no código intuitivamente, é definido o tempo de manutenção da conexão, o QOS quanto ao método de subscrição e publicação dos dados, o tópico que indicará se o cliente está online, e se será um tópico único para informar se está online ou multiplos.

```c
#define MQTT_KEEP_ALIVE_S 60

// qos passed to mqtt_subscribe
// At most once (QoS 0)
// At least once (QoS 1)
// Exactly once (QoS 2)
#define MQTT_SUBSCRIBE_QOS 1
#define MQTT_PUBLISH_QOS 1
#define MQTT_PUBLISH_RETAIN 0

// topic used for last will and testament
#define MQTT_WILL_TOPIC "/online"
#define MQTT_WILL_MSG "0"
#define MQTT_WILL_QOS 1

#ifndef MQTT_BASE_TOPIC
#define MQTT_BASE_TOPIC "rack_inteligente"
#endif

// Set to 1 to add the client name to topics, to support multiple devices using the same server
#ifndef MQTT_UNIQUE_TOPIC
#define MQTT_UNIQUE_TOPIC 1
#endif
```

Funções auxiliares

Já no cliente (Subscriber) optei por definir as funções antes da função **main()** que as chamam, assim não preciso definir as assinaturas das funções.

Primeiro defini a função **full_topic(MQTT_CLIENT_DATA_T)** que constrói a hierarquia do tópico, assim fica mais flexível quando se lida com mais de um tópico.

Já a função **control_led(bool)** é a função responsável pelo estado físico do LED controlado pelo respectivo tópico quando ocorre chama a função.

```c
static const char *full_topic(MQTT_CLIENT_DATA_T *state, const char *name) {
#if MQTT_UNIQUE_TOPIC
    static char full_topic[MQTT_TOPIC_LEN];
    snprintf(full_topic, sizeof(full_topic), "%s%s", state->mqtt_client_info.client_id, name);
    return full_topic;
#else
    return name;
#endif
}

static void control_led(bool on) {
    // Publish state on /state topic and on/off led board
    const char* message = on ? "ON" : "OFF";
    printf("Control LED: %s\n", message);
    if (on)
    {
        gpio_put(gpio: LED_RED_PIN, value: 1);
        gpio_put(gpio: LED_GREEN_PIN, value: 0);
    }
    else
    {
        gpio_put(gpio: LED_RED_PIN, value: 0);
        gpio_put(gpio: LED_GREEN_PIN, value: 1);
    }
}
```

Os Callbacks

As funções de callback são responsaveis por tratar eventos gerados pelo framework MQTT, temos três funções em questão, **pub_request_cb(void, err_t)** que trata as requisições de publicação de tópicos, e no nosso caso apenas alertamos quanto a algum erro que tenha ocorrido, sem nenhum outro tratamento.

Temos a função **sub_request_cb(void, err_t)** que trata as requisições de subscrição, primeiro ela alerta se houve algum erro, caso verdeiro emite um sinal de panico que interrompe o processamento. Caso esteja tudo bem ela processa a subscrição no tópico no qual ela foi associada somando 1 ao contador de subscrição.

Já a função **sub_request_cb(void, err_t)** que trata as requisições de subscrição, primeiro ela alerta se houve algum erro, caso verdeiro emite um sinal de panico que interrompe o processamento. Caso esteja tudo bem, ela subtrai 1 do contador de subscrição, e analisa se a contagem está correta, através do **assert** e do if que também verifica se o estado do cliente MQTT é como desligado.

```c
static void pub_request_cb(__unused void *arg, err_t err) {
    if (err != 0) {
        printf("pub_request_cb failed %d", err);
    }
}

static void sub_request_cb(void *arg, err_t err) {
    MQTT_CLIENT_DATA_T* state = (MQTT_CLIENT_DATA_T*)arg;
    if (err != 0) {
        panic(fmt: "subscribe request failed %d", err);
    }
    state->subscribe_count++;
}

static void unsub_request_cb(void *arg, err_t err) {
    MQTT_CLIENT_DATA_T* state = (MQTT_CLIENT_DATA_T*)arg;
    if (err != 0) {
        panic(fmt: "unsubscribe request failed %d", err);
    }
    state->subscribe_count--;
    assert(state->subscribe_count >= 0);

    // Stop if requested
    if (state->subscribe_count <= 0 && state->stop_client) {
        mqtt_disconnect(client: state->mqtt_client_inst);
    }
}

```

Funções Auxiliares do MQTT

A função **sub_unsub_topics(MQTT_CLIENTE_DATA_T)** é uma função que auxilia a subscrição ou descadastramento dos tópicos, associado o respetivo callback como citado acima.

```c
static void sub_unsub_topics(MQTT_CLIENT_DATA_T* state, bool sub) {
    mqtt_request_cb_t cb = sub ? sub_request_cb : unsub_request_cb;
    mqtt_sub_unsub(client: state->mqtt_client_inst, topic: full_topic(state, name: "/exit"), qos: MQT
    mqtt_sub_unsub(client: state->mqtt_client_inst, topic: full_topic(state, name: "/door"), qos: MQT
}

```

Callback que trata os dados entregues das subscrições

Temos um callback que é extremamente importante, ele é responsável por monitorar as inscrições em tópicos e tratar quando chegam dados que fora atualizados nos respectivos tópicos. A função em questão se chama **mqtt_incoming_data_cb(void,const u8_t, u16_t, u8_t)** e processa as atualizações de cada tópico, extraindo o ramo final do tópico e tratando adequadamente os dados recebidos.

```c
static void mqtt_incoming_data_cb(void *arg, const u8_t *data, u16_t len, u8_t flags) {
    MQTT_CLIENT_DATA_T* state = (MQTT_CLIENT_DATA_T*)arg;
#if MQTT_UNIQUE_TOPIC
    const char *basic_topic = state->topic + strlen(state->mqtt_client_info.client_id);
#else
    const char *basic_topic = state->topic;
#endif
    strncpy(state->data, (const char *)data, len);
    state->len = len;
    state->data[len] = '\0';

    printf("Topic: %s, Basic Topic: %s, Message: %s\n", state->topic, basic_topic, state->data);
    if (strcmp(basic_topic, "/door") == 0)
    {
        printf("Door topic: %s\n", state->data);
        if (lwip_stricmp(str1: (const char *)state->data, str2: "ON") == 0 || strcmp((const char *)s
            control_led(on: true);
        else if (lwip_stricmp(str1: (const char *)state->data, str2: "OFF") == 0 || strcmp((const ch
            control_led(on: false);
    } else if (strcmp(basic_topic, "/exit") == 0) {
        state->stop_client = true; // stop the client when ALL subscriptions are stopped
        sub_unsub_topics(state, sub: false); // unsubscribe
    }
}
```

Callback de conexão

```c
static void mqtt_connection_cb(mqtt_client_t *client, void *arg, mqtt_connection_status_t status) {
    MQTT_CLIENT_DATA_T* state = (MQTT_CLIENT_DATA_T*)arg;
    if (status == MQTT_CONNECT_ACCEPTED) {
        state->connect_done = true;
        sub_unsub_topics(state, sub: true); // subscribe;

        // indicate online
        if (state->mqtt_client_info.will_topic) {
            mqtt_publish(client: state->mqtt_client_inst, topic: state->mqtt_client_info.will_topic,
        }
    } else if (status == MQTT_CONNECT_DISCONNECTED) {
        if (!state->connect_done) {
            panic(fmt: "Failed to connect to mqtt server");
        }
    }
    else {
        panic(fmt: "Unexpected status");
    }
}
```

Função auxiliar para inicialização da conexão com o broker MQTT

```
180 static void start_client(MQTT_CLIENT_DATA_T *state) {
181 #if LWIP_ALTCP && LWIP_ALTCP_TLS
182     const int port = MQTT_TLS_PORT;
183     printf("Using TLS\n");
184 #else
185     const int port = MQTT_PORT;
186     printf("Warning: Not using TLS\n");
187 #endif
188
189     state->mqtt_client_inst = mqtt_client_new();
190     if (!state->mqtt_client_inst) {
191         panic(fmt: "MQTT client instance creation error");
192     }
193     printf("IP address of this device %s\n", ipaddr_ntoa(addr: &(netif_list->ip_addr)));
194     printf("Connecting to mqtt server at %s\n", ipaddr_ntoa(addr: &state->mqtt_broker_address));
195
196     cyw43_arch_lwip_begin();
197     if (mqtt_client_connect(client: state->mqtt_client_inst, ipaddr: &state->mqtt_broker_address, pc
198         panic(fmt: "MQTT broker connection error");
199     }
200     printf("Connected to mqtt server\n");
201 #if LWIP_ALTCP && LWIP_ALTCP_TLS
202     // This is important for MBEDTLS_SSL_SERVER_NAME_INDICATION
203     mbedtls_ssl_set_hostname(altcp_tls_context(state->mqtt_client_inst->conn), MQTT_BROKER);
204 #endif
205     mqtt_set_inpub_callback(client: state->mqtt_client_inst, pub_cb: mqtt_incoming_publish_cb, data_
206     printf("Subscribed to topics\n");
207     cyw43_arch_lwip_end();
208 }
209
```

Função de callback do DNS

```
210 // Call back with a DNS result
211 static void dns_found(const char *hostname, const ip_addr_t *ipaddr, void *arg) {
212     MQTT_CLIENT_DATA_T *state = (MQTT_CLIENT_DATA_T*)arg;
213     if (ipaddr) {
214         state->mqtt_broker_address = *ipaddr;
215         start_client(state);
216     } else {
217         panic(fmt: "dns request failed");
218     }
219 }
220
```

Primeira parte da função main(), inicialização do firmware

```c
int main(void) {
    stdio_init_all();

    // Aguarda a conexão da serial
    while (!stdio_usb_connected())
    {
        sleep_ms(ms: 1000);
    }

    gpio_init(gpio: LED_RED_PIN);
    gpio_init(gpio: LED_GREEN_PIN);
    gpio_set_dir(gpio: LED_RED_PIN, out: GPIO_OUT);
    gpio_set_dir(gpio: LED_GREEN_PIN, out: GPIO_OUT);

    printf("mqtt client starting\n");

    static MQTT_CLIENT_DATA_T state;

    if (cyw43_arch_init()) {
        panic(fmt: "Failed to inizialize CYW43");
    }
```

Função **main()** inicialização do método de conexão

Neste estágio é inicializado a metodologia de conexão ao MQTT, o uso do TLS ou SSL

```
    // Generate a unique name, e.g. pico1234
    char client_id_buf[sizeof(MQTT_BASE_TOPIC)];
    memcpy(&client_id_buf[0], MQTT_BASE_TOPIC, sizeof(MQTT_BASE_TOPIC) - 1);
    client_id_buf[sizeof(client_id_buf) - 1] = 0;
    printf("Device name %s\n", client_id_buf);

    state.mqtt_client_info.client_id = client_id_buf;
    state.mqtt_client_info.keep_alive = MQTT_KEEP_ALIVE_S; // Keep alive in sec
#if defined(MQTT_USERNAME) && defined(MQTT_PASSWORD)
    state.mqtt_client_info.client_user = MQTT_USERNAME;
    state.mqtt_client_info.client_pass = MQTT_PASSWORD;
#else
    state.mqtt_client_info.client_user = NULL;
    state.mqtt_client_info.client_pass = NULL;
#endif
    static char will_topic[MQTT_TOPIC_LEN];
    strncpy(will_topic, full_topic(state: &state, name: MQTT_WILL_TOPIC), sizeof(will_topic));
    state.mqtt_client_info.will_topic = will_topic;
    state.mqtt_client_info.will_msg = MQTT_WILL_MSG;
    state.mqtt_client_info.will_qos = MQTT_WILL_QOS;
    state.mqtt_client_info.will_retain = true;
#if LWIP_ALTCP && LWIP_ALTCP_TLS
    // TLS enabled
#ifdef MQTT_CERT_INC
    static const uint8_t ca_cert[] = TLS_ROOT_CERT;
    static const uint8_t client_key[] = TLS_CLIENT_KEY;
    static const uint8_t client_cert[] = TLS_CLIENT_CERT;
    // This confirms the indentity of the server and the client
    state.mqtt_client_info.tls_config = altcp_tls_create_config_client_2wayauth(ca_cert, sizeof(ca_
            client_key, sizeof(client_key), NULL, 0, client_cert, sizeof(client_cert));
#if ALTCP_MBEDTLS_AUTHMODE != MBEDTLS_SSL_VERIFY_REQUIRED
    WARN_printf("Warning: tls without verification is insecure\n");
#endif
#else
    state->client_info.tls_config = altcp_tls_create_config_client(NULL, 0);
    WARN_printf("Warning: tls without a certificate is insecure\n");
#endif
#endif
```

```
    cyw43_arch_enable_sta_mode();
    if (cyw43_arch_wifi_connect_timeout_ms(ssid: WIFI_SSID, pw: WIFI_PASSWORD, auth: CYW43_AUTH_WPA2_
        panic(fmt: "Failed to connect");
    }
    printf("\nConnected to Wifi\n");

    // We are not in a callback so locking is needed when calling lwip
    // Make a DNS request for the MQTT server IP address
    cyw43_arch_lwip_begin();
    int err = dns_gethostbyname(hostname: MQTT_BROKER, addr: &state.mqtt_broker_address, found: dns_
    cyw43_arch_lwip_end();

    if (err == ERR_OK) {
        // We have the address, just start the client
        start_client(state: &state);
    } else if (err != ERR_INPROGRESS) { // ERR_INPROGRESS means expect a callback
        panic(fmt: "dns request failed");
    }
```

Função main() - Super Loop

Finalmente temos o super loop que mantém a conexão ativa e aguarda por eventos de dados por até 10000ms (dez mill milisegundos)

```
    while (!state.connect_done || mqtt_client_is_connected(client: state.mqtt_client_inst)) {
        cyw43_arch_poll();
        cyw43_arch_wait_for_work_until(until: make_timeout_time_ms(ms: 10000));
    }
```

As configurações do projeto, CMakeFile.txt

O arquivo CmakeFile.txt é responsável por gerar o ambiente e orquestrado para a compilação adequada do firmware.

Vamos analisar apenas um dos CmakeFile.txt o que está mais complexo.

Configurações do ambiente

Nesta etapa inicial o ambiente é preparado, sendo feita a definição da placa que será usada, e qual SDK e Ferramentas de compilação serão usados.

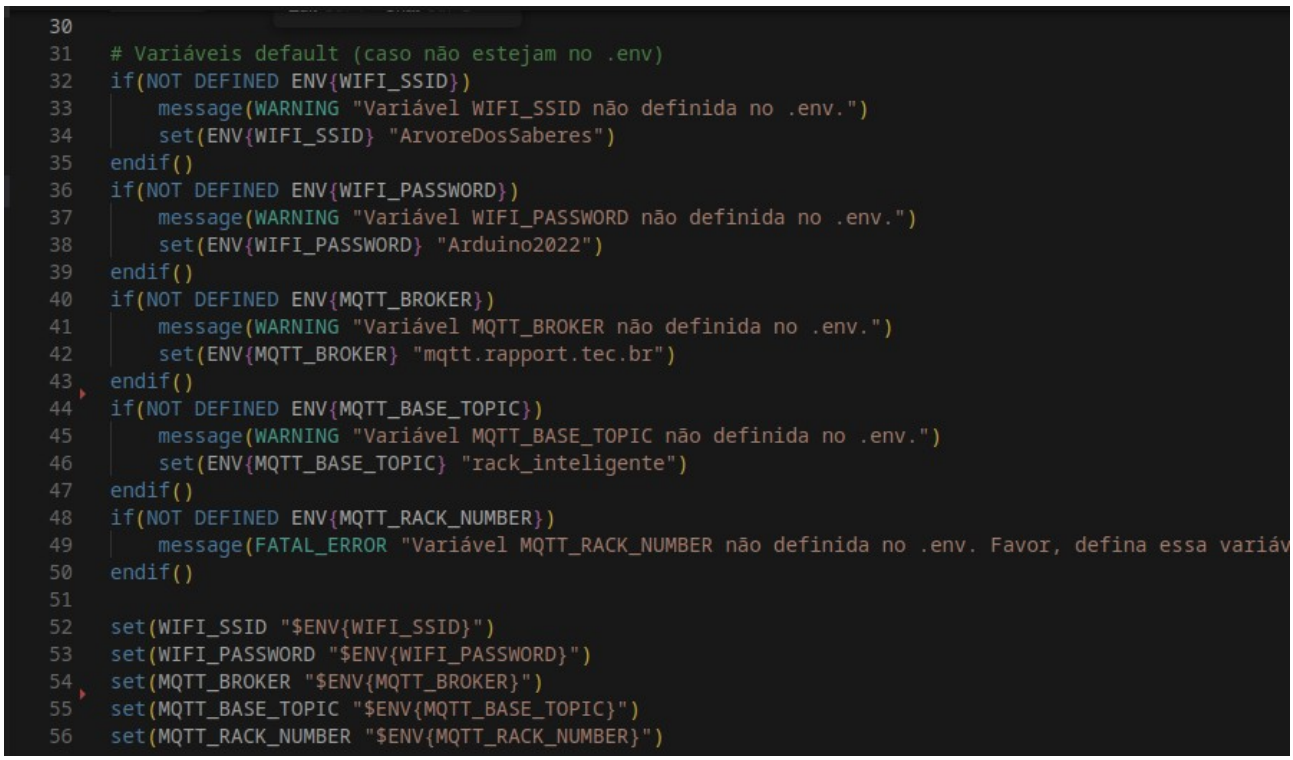
```
# == DO NOT EDIT THE FOLLOWING LINES for the Raspberry Pi Pico VS Code Extension to work ==
if(WIN32)
    set(USERHOME $ENV{USERPROFILE})
else()
    set(USERHOME $ENV{HOME})
endif()
set(sdkVersion 2.1.1)
set(toolchainVersion 14_2_Rel1)
set(picotoolVersion 2.1.1)
set(picoVscode ${USERHOME}/.pico-sdk/cmake/pico-vscode.cmake)
if (EXISTS ${picoVscode})
    include(${picoVscode})
endif()
# ====================================================================================
set(PICO_BOARD pico_w CACHE STRING "Board type")

# CMake configuration for firmware_client_mqtt
cmake_minimum_required(VERSION 3.13)

set(CMAKE_C_STANDARD 11)
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_EXPORT_COMPILE_COMMANDS ON)
```

Parametrizando a conexão

A seguir parametrizamos as macros de conexão com valores padrões e obitidas no arquivo env.cmaker



```
30
31  # Variáveis default (caso não estejam no .env)
32  if(NOT DEFINED ENV{WIFI_SSID})
33      message(WARNING "Variável WIFI_SSID não definida no .env.")
34      set(ENV{WIFI_SSID} "ArvoreDosSaberes")
35  endif()
36  if(NOT DEFINED ENV{WIFI_PASSWORD})
37      message(WARNING "Variável WIFI_PASSWORD não definida no .env.")
38      set(ENV{WIFI_PASSWORD} "Arduino2022")
39  endif()
40  if(NOT DEFINED ENV{MQTT_BROKER})
41      message(WARNING "Variável MQTT_BROKER não definida no .env.")
42      set(ENV{MQTT_BROKER} "mqtt.rapport.tec.br")
43  endif()
44  if(NOT DEFINED ENV{MQTT_BASE_TOPIC})
45      message(WARNING "Variável MQTT_BASE_TOPIC não definida no .env.")
46      set(ENV{MQTT_BASE_TOPIC} "rack_inteligente")
47  endif()
48  if(NOT DEFINED ENV{MQTT_RACK_NUMBER})
49      message(FATAL_ERROR "Variável MQTT_RACK_NUMBER não definida no .env. Favor, defina essa variáv
50  endif()
51
52  set(WIFI_SSID "$ENV{WIFI_SSID}")
53  set(WIFI_PASSWORD "$ENV{WIFI_PASSWORD}")
54  set(MQTT_BROKER "$ENV{MQTT_BROKER}")
55  set(MQTT_BASE_TOPIC "$ENV{MQTT_BASE_TOPIC}")
56  set(MQTT_RACK_NUMBER "$ENV{MQTT_RACK_NUMBER}")
```

Parametrização final

Na parametrização final é carregado o arquivo auxiliar do sdk do RP Pico, define as linguagens que serão usadas para compilar o projeto, no caso C, C++ e Asembly. Define-se o nome do executável gerado e sua versão, dsesativa a porta serial e ativa porta serial USB. Carrega as bibliotecas necessárias. Define os diretórios de inclusão de headers e finalmente transfere as macros para o estágio de compilação para que sejam visíveis internamente no código C/C++

```
set(MQTT_RACK_NUMBER "${MQTT_RACK_NUMBER}" CACHE INTERNAL "MQTT rack number for examples")

# Pull in Raspberry Pi Pico SDK (must be before project)
include(pico_sdk_import.cmake)

project(firmware_client_mqtt C CXX ASM)

# Initialise the Raspberry Pi Pico SDK
pico_sdk_init()

add_executable(firmware_client_mqtt
    firmware_client_mqtt.c
)

pico_set_program_name(firmware_client_mqtt "firmware_client_mqtt")
pico_set_program_version(firmware_client_mqtt "0.1")

pico_enable_stdio_uart(firmware_client_mqtt 0)
pico_enable_stdio_usb(firmware_client_mqtt 1)

target_link_libraries(firmware_client_mqtt
    pico_stdlib
    pico_cyw43_arch_lwip_threadsafe_background
    pico_lwip_mqtt
)

target_include_directories(firmware_client_mqtt PRIVATE ${CMAKE_CURRENT_LIST_DIR})

# Wi-Fi and MQTT configuration (edit as needed)
target_compile_definitions(firmware_client_mqtt PRIVATE
    WIFI_SSID=\"${WIFI_SSID}\"
    WIFI_PASSWORD=\"${WIFI_PASSWORD}\"
    MQTT_BROKER=\"${MQTT_BROKER}\"
    MQTT_BASE_TOPIC=\"${MQTT_BASE_TOPIC}\"
    MQTT_RACK_NUMBER=\"${MQTT_RACK_NUMBER}\"
)

pico_add_extra_outputs(firmware_client_mqtt)
```