

Obiekty Internetu Rzeczy

(Laboratorium 2)

Krzysztof Pierczyk

listopad 2020

1 Wstęp

W ramach drugiego ćwiczenia laboratoryjnego stworzony został prosty węzeł IoT komunikujący się za pomocą protokołu MQTT. Projekt zrealizowano w oparciu o płytkę rozwojową NodeMCUv3 z układem SoC ESP8266. Dzięki dostarczonemu przez producenta środowisku programistycznemu [1] możliwe było wykorzystanie gotowej implementacji protokołu w postaci otwartoźródłowej biblioteki `esp-mqtt` [2]. Utworzony program pracuje pod kontrolą popularnego systemu operacyjnego **FreeRTOS**.

2 Opis rozwiązania

Biblioteka `esp-mqtt` udostępnia niezmiernie proste API pozwalające na komunikację w pełni zgodną ze standardem MQTT 3.1.1. Jej wykorzystanie sprowadza się do zaimplementowania dwóch podstawowych elementów: struktury inicjalizacyjnej oraz zestawu funkcji typu *callback*. Pierwszy z nich zaprezentowany został na poniższym listingu.

```
esp_mqtt_client_config_t mqtt_cfg = {
    // Connection config
    .uri = "mqtt://192.168.0.94:1883",
    .client_id = "esp8266",
    // Session parameters
    .disable_clean_session = false,
    .keepalive = 5,
    // Last Will & Testament
    .lwt_topic = "esp8266/status",
    .lwt_msg = "disconnected",
    .lwt_qos = 1,
    .lwt_retain = true,
};

// Initialize the MQTT client using the defined configuration
esp_mqtt_client_handle_t client = esp_mqtt_client_init(&mqtt_cfg);
// Register handlers for library's events
mqtt_register_events_handlers(client);
// Run the client
esp_mqtt_client_start(client);
```

Każda instancja klasy `esp_mqtt_client_config_t` opisuje pojedyncze połączenie z brokerem. Zawiera ona parametry umożliwiające kontrolę wszystkich aspektów komunikacji. Na listingu przedstawione zostały tylko te pola struktury, których wartości domyślne nie odpowiadały założeniom projektowym. Pierwsze z nich zawiera URI brokera. W tym przypadku jest to adres i port, na którym nasłuchuje lokalny serwer Mosquitto. Następne pole konfiguruje identyfikator klienta umieszczany w wiadomości `CONNECT`.

W kontekście sesji możliwe jest ustalenie czasu `keepalive`, który determinuje jak długo połączenie będzie utrzymywane przy braku ruchu pomiędzy klientem a serwerem. W tym przypadku został on

ustalony na 5 sekund, aby możliwe było przetestowanie działania mechanizmu *Last Will & Testament*. Każde połączenie będzie zestawiane jako czysta sesja, co konfiguruje pole `disable_clean_session`. Sprawi to, że wiadomości wysyłane na tematy subskrybowane przez węzeł nie będą kolejgowane na serwerze w przypadku rozłączenia się klienta.

Ostatnie cztery pola konfiguruja zachowania brokera w sytuacji niespodziewanego zerwania połączenia. W naszym przypadku na temat `esp8266/status` wysłana zostanie (z poziomem *QoS* = 1 oraz znacznikiem *Retain* = *true*) wiadomość "disconnected". W połączeniu z wiadomością "connected", wysyłaną przez węzeł zaraz po zestawieniu połączenia, stworzy to mechanizm umożliwiający rozpoznanie jego stanu przez innych klientów MQTT.

```
static void mqtt_event_handler_data(  
    void *handler_args,  
    esp_event_base_t base,  
    int32_t event_id,  
    void *event_data  
) {  
    ESP_LOGI(TAG, "MQTT_EVENT_DATA");  
  
    esp_mqtt_event_handle_t event =  
        (esp_mqtt_event_handle_t) event_data;  
  
    printf("Received from topic:%.s. Data is:\n%.s\r\n",  
        event->topic_len, event->topic,  
        event->data_len, event->data  
    );  
}
```

Tak przygotowana struktura wykorzystywana jest do stworzenia reprezentacji klienta MQTT przez bibliotekę. Dla każdego klienta należy zarejestrować zestaw funkcji *callback*. Każda z nich zostaje wywołana przez bibliotekę w momencie wystąpienia określonego zdarzenia (np. błąd komunikacji, zestawienie połączenia, otrzymanie danych). Przykładowa funkcja została przedstawiona na powyższym listingu. Po skonfigurowaniu klienta oraz zarejestrowaniu funkcji należy zainicjalizować połączenie poprzez wywołanie `esp_mqtt_client_start(client)`. Należy zauważyć, że funkcja ta tworzy odrębny wątek w systemie, który zajmuje się analizą oraz delegowaniem przychodzących i wychodzących pakietów.

Po wykonaniu powyższych kroków wątek główny może przystąpić do subskrybowania oraz publikowania wiadomości na wybrane tematy. W tym przypadku węzeł rozpoczyna od wysłania w.w. informacji o statusie oraz zasubskrybowania przykładowego tematu **temperature**. Informacje o otrzymywanych danych będą wysyłane poprzez port szeregowy. Ostatecznie, wątek rozpoczyna nieskończoną pętlę, w której co sekundę publikuje na temat `esp8266/heap` informacje o ilości wolnej pamięci na sterce.

2.1 Komunikacja poprzez TLS

Biblioteka `esp-mqtt` umożliwia także komunikację z wykorzystaniem protokołu TLSv1.2 (implementacja oparta o `mbedtls` [3]). Na chwilę obecną autoryzacja możliwa jest tylko z wykorzystaniem certyfikatów. W ramach zadania zrealizowana została dodatkowa wersja wyżej przedstawionego klienta wykorzystująca protokół szyfrujący. Przygotowanie odpowiedniego kanału komunikacyjnego składało się z kilku kroków:

- przygotowania pary kluczy asymetrycznych oraz certyfikatu dla tymczasowego CA (ang. *Certificate Authority*)
- przygotowania pary kluczy asymetrycznych dla serwera Mosquitto
- zarejestrowanie certyfikatu serwera z wykorzystaniem utworzonego CA
- zapisanie certyfikatu CA w pamięci układu SoC

- skonfigurowanie Mosquitto do działania z wykorzystaniem utworzonych kluczy oraz certyfikatu

Wszystkie operacje zostały przeprowadzone przy użyciu narzędzia **openssl**. Binarny obraz certyfikatu CA został wbudowany w strukturę oprogramowania węzła jako tablica danych całkowitoliczbowych. Została ona przekazana do biblioteki **esp-mqtt** w procesie konfiguracji, co umożliwiło autoryzację serwera w czasie nawiązywania połączenia.

3 Testowanie rozwiązania

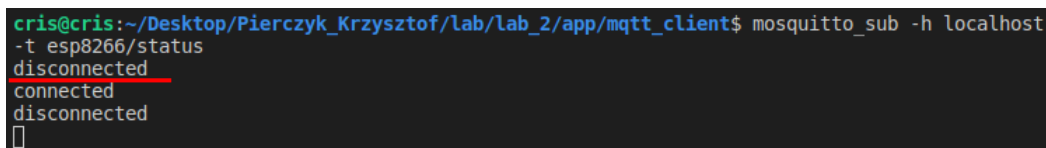
Jak już wspomniano układ ESP8266 zestawiał połączenie z brokerem **Mosquitto** działającym na komputerze dołączonym do sieci LAN. Serwer ten nasłuchiwał na standardowych portach 1883 (połączenie nieszyfrowane) oraz 8883 (połączenie szyfrowane). Do testowania poprawności komunikacji wykorzystano również narzędzia **mosquitto_sub** oraz **mosquitto_pub** z pakietu mosquitto. Przeprowadzono po trzy testy dla każdej z wersji programu (z i bez TLS):

- sprawdzenie informacji publikowanych na temacie esp8266/status
- sprawdzenie informacji publikowanych na temacie esp8266/heap
- sprawdzenia informacji otrzymywanych przez węzeł z tematu temperature

Każdy ze scenariuszy testowych został zarejestrowany z wykorzystaniem pakietu wireshark.

3.1 Publikowanie na temat esp8266/status

W pierwszym scenariuszu testowym, klient MQTT zasubskrybował kanał esp8266/status po uwczesnym odłączeniu modułu ESP8266 od zasilania. Dzięki temu ostatnia wiadomość opublikowana na tym kanale (utrzymywana ze względu na obecność flagi *Retain* w pakiecie **CONNECT**) powinna mieć wartość "disconnected". Po ponownym podłączeniu węzła łączy się on z serwerem i aktualizuje swój status. Taka sytuacja została przedstawiona na Rys. 1.



```
cris@cris:~/Desktop/Pierczyk_Krzysztof/lab/lab_2/app/mqtt_client$ mosquitto_sub -h localhost
-t esp8266/status
disconnected
connected
disconnected
█
```

Rysunek 1: Wiadomości otrzymywane przez klienta subskrybującego temat esp8266/status przed i po uruchomieniu węzła (moment uruchomienia oznaczono czerwoną linią)

3.2 Publikowanie na temat esp8266/heap

W przypadku drugiego testu pomocniczy klient obserwował wiadomości pojawiające się na temacie esp8266/heap. Zgodnie z oczekiwaniami węzeł publikował na nim ilość dostępnej pamięci (Rys. 2). Warto zauważyć, że ilość wykorzystywanej przez wątek MQTT pamięci ustala się dopiero po pewnej chwili od uruchomienia urządzenia, co skutkuje odmienną wartością pierwszej próbki.

3.3 Otrzymywanie wiadomości z tematu temperature

Ostatni z testów polegał na wysłaniu na topic temperature kilku wiadomości z wykorzystaniem i zaobserwowaniu ich odebrania przez węzeł (został on skonfigurowany tak, aby wyświetlać informacje o danych przychodzących z serwera na porcie szeregowym). Rezultat został przedstawiony na Rys. 3.

Literatura

- [1] https://github.com/espressif/ESP8266_RTOS_SDK
- [2] <https://github.com/espressif/esp-mqtt>
- [3] <https://github.com/espressif/mbedtls>