OBIR - Dokumentacja

* Stosowane technologie
* Język programowania C/C++
* Środowisko Arduino
* Biblioteka własna CoAP będąca dopasowaną do potrzeb projektu wersją biblioteki „Simple coap library” (<https://github.com/hirotakaster/CoAP-simple-library>)
* Działanie CoAP-a

Działanie naszego serwera opiera się na funkcji CoAP::loop(), która jest za odpowiedzialna za sprawdzanie czy na słuchanym porcie otrzymaliśmy nowy pakiet UDP. Gdy pojawi się nowa wiadomość serwer sprawdza czy potrafi obsłużyć żądanie na podane URI. Jeśli serwer nie może znaleźć podanego URI zwraca wiadomość NOT FOUND. W przypadku pomyślnego odnalezienia URI, żądanie obsługuje odpowiednia funkcja callback(CoAPPacket & packet, IPAdress ip, int port). Funkcje callback definiuje się w implementacji serwera, a następnie żeby serwer mógł użyć zdefiniowanej funkcji należy ją dodać serwera za pomocą CoAP::server(callback c, String uri). Aby umożliwić obserwowanie zasobów stworzyliśmy strukturę Obserwer, która pozwala zapisać dane adresowe obserwatora i token, aby móc potem wysyłać mu powiadomienia o zmianach w obserwowanym zasobie. Serwer również przetrzymuje stan każdego zasobu łącznie i przypisanego mu tagu. Pozwala to korzystać z mechanizmu Etag.

* Komunikacja radiowa

Do zaimplementowania komunikacji radiowej wykorzystaliśmy bibliotek:

-RF24Network.h

-RF24.h

Komunikacja radiowa odbywa się na kanale 65.

ID węzła serwera używane do komunikacji radiowej : 00

ID węzła MiniPro używane do komunikacji radiowej: 01

Wiadomości wymieniane między węzłami za pomocą komunikacji radiowej zawierają:

- czas w milisekundach od uruchomienia węzła nadawcy

- typ zasobu, którego dotyczy wiadomość:

0 – wszystkich

1 – lampka

2 – klawiatura

- wartość zasobu lub typ żądania w zależności od zasobu i nadawcy wiadomości:

wszystkich:

‘g’ - get

lampka:

‘g’ - get

‘0’ - wyłącz / jest wyłączona

‘1’ - włącz / jest włączona

klawiatura:

‘g’ - get

od ‘0’ do ’9’ oraz ‘\*’ i ‘#’ - odpowiedni klawisz klawiatury

Aby stan zasobów był znany bez względu na kolejność włączenia węzłów, po włączeniu któregoś z nich wysyłane są odpowiednie wiadomości. Następnie w funkcji loop() węzeł oczekuje na przychodzącą wiadomość i w wypadku jej pojawienia się odpowiednio ją przetwarza. Węzeł serwera jest informowany przez węzeł MiniPro o zmianie stanu klawiatury za każdym razem kiedy to nastąpi.

* Sposób testowania systemu i wykonane testy

W celu właściwego przetestowania systemu zaimplementowane zostały odpowiednie komunikaty, informujące o wykonaniu się w bloków kodu. Można je obserwować wykorzystując Serial Monitor.

Po wykonaniu zapytań obserwowaliśmy wszystkie komunikaty i zgodność z założeniami projektowymi.

* Lista funkcjonalności i scenariusz testowy

|  |  |
| --- | --- |
| Funkcjonalności | |
| Obsługa wiadomości NON – GET | Test 1, Test 2, Test 3, Test 4 |
| Obsługa wiadomości NON – PUT | Test 2 |
| Obsługa opcji Content-Format | Test 1, Test 2, Test 3, Test 4 |
| Obsługa opcji Uri-path | Test 1, Test 2, Test 3, Test 4 |
| Obsługa opcji Accept | Test 2 |
| Obsługa tokena | Test 2, Test 3, |
| Obsługa MID | Test 1, Test 2, Test 3, Test 4 |
| Obsługa opcji Observe | Test 3 |
| Obsługa opcji ETag | Test 2, Test 3, Test 4 |
| Zasób opisujący pozostałe zasoby (GET) | Test 1 |
| Lampka (GET i PUT) | Test 2 |
| Klawiatura – zasób obserwowalny (GET) | Test 3 |
| Zestaw metryk (GET) | Test 4 |

Test 1: .well-known/core – użycie funkcji discover

Test 2: Lampka – Użycie funkcji Get z tokenem na zasobie light, użycie funkcji GET z użytym wcześniej ETagiem (Odpowiedź powinna być ACK-2.03 Valid), użycie wiadomości PUT z wartością 1 (włączenie lampki), ponownie użycie funkcji GET z użytym wcześniej ETagiem (ACK-2.05 Content)

Test 3: Klawiatura – Użycie funkcji GET z tokenem na zasobie keyboard, rozpoczęcie obserwowania zasobu keyboard, kilkukrotne wciśnięcie różnych przycisków na klawiaturze.

Test 4: Zestaw metryk – Użycie funkcji GET z tokenem na zasobie statistics