Biblioteka komunikacji międzysystemowej

Autor: Miłosz Łagan

Spis treści

1. Wstęp

Biblioteka została stworzona w celu ułatwienia obsługi nowego standardu komunikacji międzysystemowej w projekcie łazika planetarnego w KN AGH Space Systems. Głównym zadaniem implementacji, jako przenośnego kodu dystrybuowanego w formie biblioteki jest standaryzacja mechanizmów obsługi przychodzących i wychodzących pakietów w autorskim stosie komunikacyjnym, zmniejszając czas potrzebny na implementację nowych rozwiązań pracujących w tym systemie oraz redukcja ilości potencjalnych błędów.

2. Założenia

Podczas przygotowywania projektu postawiono następujące założenia:

- Możliwość zastosowania biblioteki pod systemami *Windows / Linux* oraz w środowiskach mikrokontrolerów dysponujących średnią wielkością pamięci wbudowanej (32KB+)
- Generyczne przekazywanie protokołów wyższych warstw
- Łatwa możliwość rozszerzenia funkcjonalności w przyszłości
- Wiele sposobów na przekazywanie danych do biblioteki - parsowanie dyskretnych wiadomości, użycie predefiniowanych buforów wejściowo-wyjściowych lub zastosowanie niestandardowych struktur danych zdefiniowanych przez użytkownika
- Użycie biblioteki *ETL*[1] zastępującej bibliotekę standardową - statyczna alokacja pamięci
- Brak użycia dynamicznej alokacji pamięci ze względu na docelowe zastosowanie w systemach wbudowanych
- Brak obsługi wyjątków - przekazywanie błędów za pomocą typu wyliczeniowego

[1] Embedded Template Library - https://www.etlcpp.com/

3. Projekt techniczny

Biblioteka docelowo będzie wspierać nowo zaprojektowany stos protokołów komunikacyjnych. Dzieli się on na trzy warstwy, oraz ideowo przypomina stos TCP/IP. Wszystkie z nich operują na danych typu binarnego.

Warstwa	Rola
Aplikacji	Wymiana danych
Transportowa	Adresacja, gwarancja dostarczenia, fragmentacja, enkapsulacja
Łącza	Synchronizacja, framing

Tabela 1. Role warstw protokołu

Warstwa łącza jest zapewniona przez fizyczny, sprzętowy nadajnik i odbiornik i jest niezależna od reszty stosu - w ramach potrzeby istnieje możliwość zmiany medium transmisyjnego na inne bez ingerencji w wyższe warstwy komunikacji. Dane będą mogły być wymieniane przez tworzony system radiowy oparty na technologii *LoRa*[2] w paśmie 434 MHz oraz 2.4 GHz, jak i zwykły protokół szeregowy *UART*[3] popularny w mikrokontrolerach

Biblioteka obsługuje warstwę transportową oraz częściowo aplikacji na przestrzeni zdefiniowanych w kodzie protokołów warstwy aplikacji.

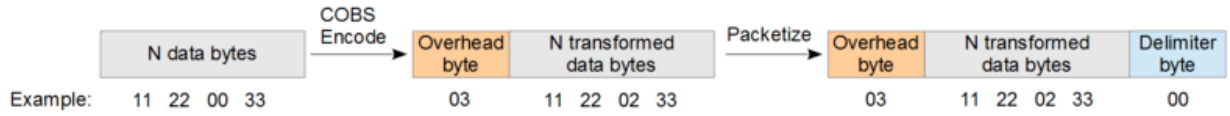
Warstwa	Protokoły
Aplikacji	Kalman Binary Protocol (KBX), Link Control Protocol (LCP), Raw Data Stream (RDS)
Transportowa	Frame Transport Protocol (FTP)
Łącza	*LoRa MAC, UART*

Tabela 2. Protokoły

Głównym protokołem jest *Frame Transport Protocol* umożliwiający enkapsulację wyższych warstw bez znajomości ich zawartości. Jest to szczególnie przydatna właściwość systemu, ponieważ pozwala na przekazywanie danych między poszczególnymi urządzeniami bez wiedzy o typie danych. Wyżej wymieniony protokół składa się z nagłówka, wiadomości warstwy wyższej oraz pola *CRC16*[4]. Całość jest dodatkowo kodowana za pomocą algorytmu

[2] LoRa - https://www.semtech.com/lora/
[3] Universal asynchronous receiver-transmitter - https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter
[4] Cyclic redundancy check

COBS - Consistent Overhead Byte Stuffing[5]. Umożliwia on wykluczenie z puli 256 symboli jednego z nich w celu zastosowania go do rozgraniczenia ramek. Wiąże się to z dodaniem nadmiarowego bajtu występującego co najwyżej raz na 254 znaki. W porównaniu do innych mechanizmów ramkowania (*HDLC*, *PPP*), których narzut jest zmienny od danych wprowadzanych do systemu, stała ilość nadmiarowych bajtów jest pożądaną zaletą szczególnie w systemach mikroprocesorowych.



Rysunek 1. Schemat kodowania COBS (CC-BY-SA-4.0 Lambtron)

Pole nagłówka	Rozmiar	Opis
Destination node	3b adres, 5b port	Usługa docelowa pakietu
Source node	3b adres, 5b port	Usługa źródłowa pakietu
Delivery type + Priority	1B	Typ dostarczania danych oraz priorytet
Protocol ID	1B	Numer protokołu warstwy wyższej
Payload length	1B	Długość wiadomości warstwy wyższej

Tabela 2. Nagłówek protokołu FTP

W celu zapewnienia integralności danych użyto algorytmu *CRC16*, który umożliwia weryfikację poprawności odebranych danych oraz odrzucenie wiadomości w przypadku wystąpienia zakłóceń na łączu. Jest to dodatkowa warstwa zabezpieczająca przed interpretacją niepożądanego pakietu - w przypadku połączenia radiowego warstwa łącza danych także posiada sumę kontrolną oraz kod korekcyjny.

[5] http://www.stuartcheshire.org/papers/COBSforToN.pdf

4. Opis realizacji

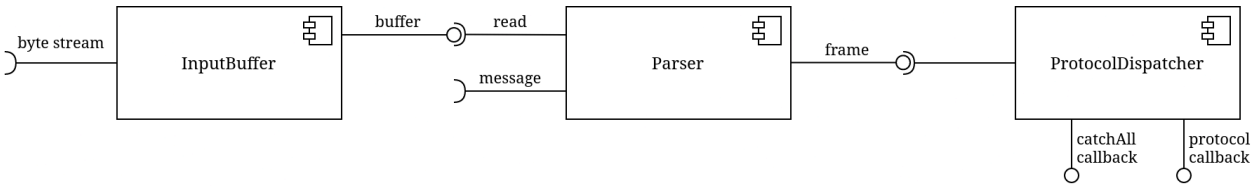
4.1 Zastosowane technologie

Kod jest dystrybuowany w postaci biblioteki używającej do kompilacji systemu *CMake*. Projekt został napisany w języku C++ w wersji 20. W celu umożliwienia efektywnej i bezpiecznej implementacji w systemach wbudowanych zamiast biblioteki standardowej użyto *ETL - Embedded Template Library*. Dostarcza ona kolekcję kontenerów, algorytmów i narzędzi nie używających dynamicznej alokacji pamięci, zoptymalizowanych pod działanie z ograniczoną ilością zasobów.

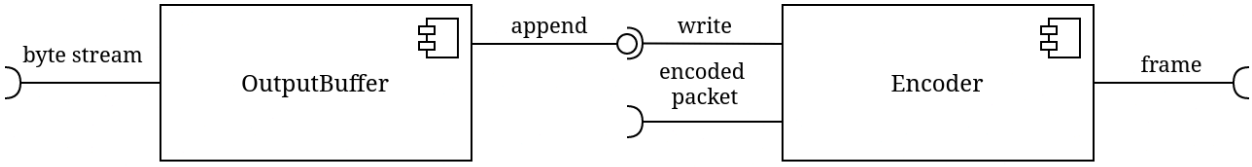
W bibliotece zawarto elementy dziedziczenia oraz szablony, a kod został pogrupowany w tematycznych folderach:

- **Comms2** - główny katalog, zawiera wspólne typy oraz definicję parsera i enkodera pakietów warstwy transportowej
- **Comms2/protocols** - katalog zawierający definicje protokołów warstwy aplikacji (**/application**) oraz szczegóły kodowania warstwy transportowej (**/datalink**)
- **Comms2/buffer** - katalog zawierający definicję interfejsu bufora wejściowego i wyjściowego oraz przykładowo zaimplementowane bufory
- **Comms2/dispatcher** - katalog zawierający mechanizm obsługi przychodzących pakietów przez warstwy wyższe oraz ich filtrowania i trasowania

4.2 Przepływ danych



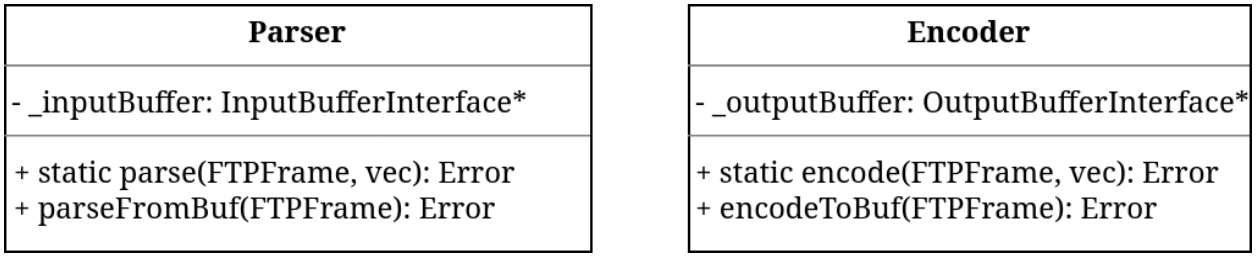
Rysunek 2. Diagram komponentów - odbieranie pakietów



Rysunek 3. Diagram komponentów - wysyłka pakietów

4.2.1 Encoder / Parser

Klasy *Encoder* i *Parser* umożliwiają na translację pakietów z danych przychodzących na łączu danych na zinterpretowany i zweryfikowany typ *FTPFrame* symbolizujący ramkę warstwy transportowej. Następuje automatyczna weryfikacja poprawności danych wejściowych, a ewentualne błędy są zwracane do użytkownika.

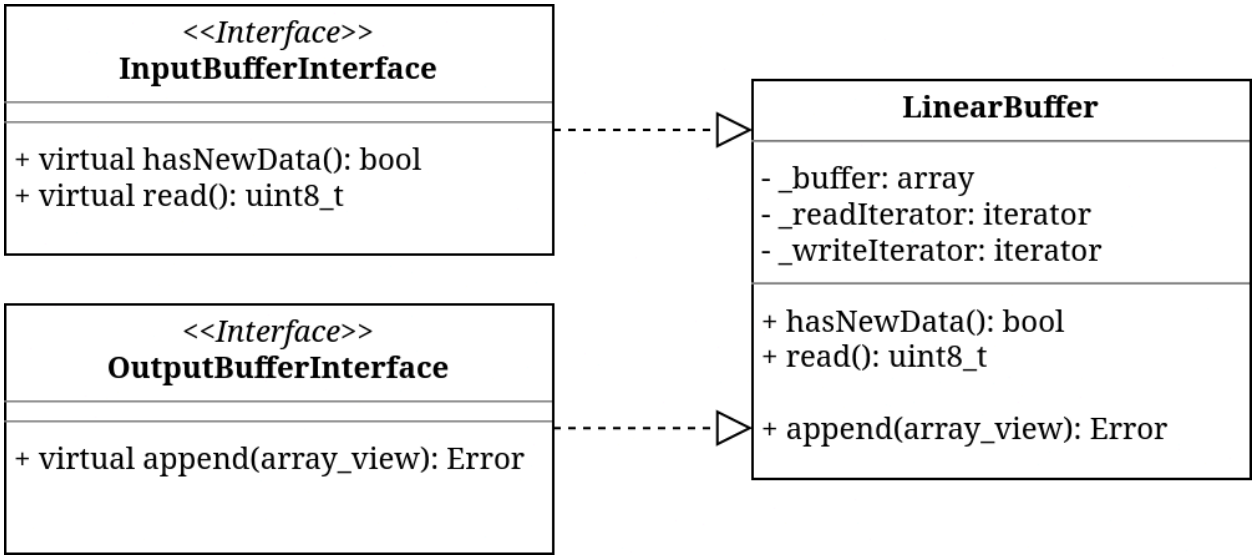


Rysunek 4. Diagram klas Parser i Encoder

Surowe dane binarne na których operują klasy są dostarczane przez bufory opisane w 4.2.2.

4.2.2 InputBuffer / OutputBuffer

Bufory pełnią rolę miejsca do przetrzymywania danych wejściowych do interpretacji przez bibliotekę, albo danych wyjściowych do dostarczenia do docelowego systemu.



Rysunek 5. Diagram klas buforów wejścia / wyjścia

Rolą klas jest dostarczenie zunifikowanego interfejsu dla Parser'a / Encoder'a. Użytkownik może skorzystać z dostarczonych implementacji, albo zaimplementować swój typ bufora, pasującego do zastosowania i platformy. Możliwe jest połączenie zarówno ze strumieniem bajtów, jak i pogrupowanymi surowymi wiadomościami.

4.2.1 Protocol Dispatcher

Rolą *ProtocolDispatcher*'a jest umożliwienie dalszego przetwarzania i przekierowywania pakietów w systemie przez logikę aplikacji klienta. W tym celu klasa umożliwia pobranie pakietu, który następnie zostanie przekierowany do właściwego protokołu komunikacyjnego warstwy aplikacji. Za pomocą architektury *callback* można zarejestrować zachowanie, jakie wywoła przyjście pakietu o określonych polach.

4.3 Testy jednostkowe

Poprzez bibliotekę *GTest* umożliwiono weryfikację poprawności zaimplementowanych warstw abstrakcji i algorytmów. W celu wywołania testu należy w folderze *build* wywołać komendę *ctest*.

Testy obejmują większość udostępnianych użytkownikowi funkcjonalności aplikacji i znajdują się w folderze *tests/* w korzeniu projektu.

4.4 Przykłady zastosowania

W celu ułatwienia implementacji biblioteki we własnych aplikacjach w folderze *examples/* zawarto przykładowe zastosowania poszczególnych funkcji udostępnianych użytkownikowi oraz prosty interfejs *CLI* pozwalający na wypróbowanie możliwości biblioteki bez konieczności tworzenia własnego kodu.