LoXiM — Specyfikacja protokołu komunikacyjnego Klient↔Serwer oraz Serwer↔Serwer

stan na 2006-10-26

Piotr Tabor (pt214569@students.mimuw.edu.pl)

4 czerwca 2008

Wersja: 0.2

Historia

Data	Wersja	Autor	Zmiany
2006-10-26	0.1	Piotr Tabor	Pierwsza wersja dokumentu
2008-06-01	0.2	Piotr Tabor	Oczyszczenie dokumentu

Spis treści

1 Wstęp

Celem tego dokumentu jest udokumentowanie protokołu sieciowego, jaki jest wykorzystywany przez semistrukturalną bazę danych „LoXiM". Konieczność stworzenia tego dokumentu wynika z faktu, że bieżący protokół komunikacyjny nie jest w żaden sposób udokumentowany, co uniemożliwia stworzenie aplikacji integrujących się z tą semistrukturalną bazą danych.

W dokumencie pojawiają się adnotacje dotyczące użycia lub jego braku - funkcji „htonl". Jest to standardowa funkcja języka C, która służy do konwersji danej liczby z architektury lokalnej komputera na liczbę kodowaną w systemie Big-endian.

W poniższym dokumencie pojawiają się też rozmiary i offsety względem początku paczki zapisane w nawiasach zwykłych. Używałem ich do oznaczenia sytuacji prawdziwej w przypadku architektury 64 bitowej (w przeciwieństwie do domyślnie przyjętej w tym dokumencie architektury 32bitowej). Oczywiście występują analogiczne różnice dla pozostałych architektur - o innym rozmiarze słowa procesora.

2 Rodzaje pakietów

SimpleQueryPackage Przesłanie prostego (bezparametrowego zapytania)

ErrorPackage Odpowiedź serwera stwierdzająca wystąpienie błędu.

ParamQueryPackage Wysłanie zapytania, które może zawierać parametry.

StatementPackage Otrzymanie od serwera ID wysłanego zapytania.

ParamStatementPackage Wysłanie parametrów do już wysłanego zapytania na serwer.

SimpleResultPackage Odpowiedź od serwera z wynikami przetwarzania zapytania.

RemoteQueryPackage Przetwarzanie rozproszone. Serwer prosi inny serwer o wykonanie podzapytania.

RemoteResultPackage Przetwarzanie rozproszone. Serwer odpowiada wynikami zapytania na zapytanie.

3 Standardowy format pakietu

O pakiecie jest wiadomo tylko jedno: zaczyna się od jednobajtowej stałej świadczącej o typie pakietu.

Od - do	Wartość	Zawartość
0 ⟷ 0		Stała określająca typ pakietu
1 ⟷ n		Dane pakietu

4 Standardowy format struktury danych

4.1 Obiekt tekstowy = Result::STRING

Przesyła łańcuch tekstu.

Od - do	Wartość	Zawartość
0 ⟷ 3(7)	Result::STRING	Stała mówiąca o typie rozważanego obiektu
4 (8) ⟷ 7 (15)	n (unsigned long)	Długość stringa (zast. fun. „htonl")z wliczonym znakiem \000
8 (16) ⟷ n+7 (n+15)	unsigned long	Przesyłany łańcuch
n+7 (n+15) ⟷ n+7 (n+15)	\000	Znak końca łańcucha

4.2 Obiekt pusty = Result::VOID

Dane puste - NULL.

Od - do	Wartość	Zawartość
0 ⟷ 3(7)	Result::VOID	Stała mówiąca o typie rozważanego obiektu

4.3 Informacja o błędzie = Result::ERROR

Przysyła unsigned long int (co z liczbami ujemnymi !!!).

Od - do	Wartość	Zawartość
0 ⟷ 3(7)	Result::ERROR	Stała mówiąca o typie rozważanego obiektu
4(8) ⟷ 7(15)	unsigned long	Numer błędu (zast. fun. „htonl")

4.4 Liczba całkowite (dodatnia) = Result::INT

Przysyła unsigned long int (co z liczbami ujemnymi !!!).

Od - do	Wartość	Zawartość
0 ⟷ 3(7)	Result::INT	Stała mówiąca o typie rozważanego obiektu
4(8) ⟷ 7(15)	unsigned long	Dana liczba (zast. fun. „htonl")

4.5 Wartość rzeczywista = Result::DOUBLE

Przesyła liczbę rzeczywistą. Liczba jest nie poprawnie konwertowana za pomocą funkcji (zast. fun. „htonl")

Od - do	Wartość	Zawartość
0 ⟷ 3(7)	Result::INT	Stała mówiąca o typie rozważanego obiektu
4(8) ⟷ 11(15)	double	Dana liczba - Nie poprawne użycie funkcji „htonl" - każde 4 bajty oddzielnie ?!

4.6 Prawda = Result::BOOLTRUE

Typ prawdy.

Od - do	Wartość	Zawartość
0 ⟷ 3(7)	Result::BOOLTRUE	Stała mówiąca o typie rozważanego obiektu
4(8) ⟷ 7(15)	1	Stała mówiąca, że to prawda (sam typ jak widać nie wystarczył autorowi tego rozwiązania)

4.7 Fałsz = Result::BOOLFALSE

Typ fałszu

Od - do	Wartość	Zawartość
0 ⟷ 3(7)	Result::BOOLFALSE	Stała mówiąca o typie rozważanego obiektu
4(8) ⟷ 7(15)	0	Stała mówiąca, że to fałsz (sam typ jak widać nie wystarczył autorowi tego rozwiązania)

4.8 Typ logiczny = Result::BOOL

Typ wartości logicznej (logika dwuwartościowa)

Od - do	Wartość	Zawartość
0 ⟷ 3(7)	Result::BOOL	Stała mówiąca o typie rozważanego obiektu
4(8) ⟷ 7(15)	0 lub 1	Stała mówiąca,czy to fałsz(0), czy prawda(1)

4.9 Łącznik = Result::BINDER

Binder - dowiązanie pomiędzy nazwą obiektu a wartością związaną z tą nazwą.

Od - do	Wartość	Zawartość
0 ⟷ 3(7)	Result::STRING	Stała mówiąca o typie rozważanego obiektu
4 (8) ⟷ 7 (15)	n (unsigned long)	Długość stringa (zast. fun. „htonl")z wliczonym znakiem \000
8 (16) ⟷ n+7 (n+15)	unsigned long	Przesyłany łańcuch
n+7 (n+15) ⟷ n+7 (n+15)	\000	Znak końca łańcucha
n+8 (n+16) ⟷ ...		Dane związane z tą nazwą. Patrz: Standardowy format struktury danych - rekurencja

4.10 Multizbiór = Result::BAG

Implementowane przez funkcję get/setResultCollection. Reprezentuje multizbiór obiektów.

Od - do	Wartość	Zawartość
0 ⟷ 3(7)	Result::BAG	Stała mówiąca o typie rozważanego obiektu
4(8) ⟷ 7(15)	unsigned int	Liczba mówiąca o liczbie obiektów w rozważanej kolekcji
8(16) ⟷ ...		Kolejne definicje poszczególnych obiektów (patrz: Standardowy format struktury danych - czyli rekurencja)

4.11 Sekwencja = Result::SEQUENCE

Tak samo jak BAG. (Różni się tylko identyfikatorem typu struktury danych w pierwszych 4 (8) bajtach: Result::SEQUENCE)

4.12 Struktura = Result::STRUCT

Tak samo jak BAG. (Różni się tylko identyfikatorem typu struktury danych w pierwszych 4 (8) bajtach: Result::STRUCT)

4.13 Referencja = Result::REFERENCE

Referencja.

Od - do	Wartość	Zawartość
0 ⟷ 3(7)	Result::REFERENCE	Stała mówiąca o typie rozważanego obiektu
4(8) ⟷ 7(15)	int	Liczba będąca logicznym ID obiektu. UWAGA: liczba ta obecnie nie jest poddawana konwersji funkcją „htonl"...!!!

4.14 Wynik = Result::RESULT

Typ nie obsługiwany przy serializacji i deserializacji - zwraca kod błędu -2.

5 Przegląd typów pakietów

5.1 SimpleQueryPackage

Pakiet tego typu służy do przesłania prostego (bezparametrowego zapytania).

Od - do	Wartość	Zawartość
0 ⟷ 0	0	Stała określająca typ pakietu
1 ⟷ n		Łańcuch określający zapytanie - nie może zawierać znaku \000
n+1 ⟷ n+1	\000	Znak końca string'a (\000) - a tym samym końca pakietu

5.2 ErrorPackage

Pakiet zawiera numer błędu, który wystąpił po stronie serwera.

Od - do	Wartość	Zawartość
0 ⟷ 0	5	Stała określająca typ pakietu
1(1) ⟷ 4(8)		Numer błędu - wielkości sizeof(int) - a więc zależny od architektury!!!

5.3 ParamQueryPackage

Wysłanie zapytania, które może zawierać parametry. W strukturze komunikat nie różni się niczym od SimpleQueryPackage (tzn. jedyną różnicą jest identyfikator typu pakietu).

Od - do	Wartość	Zawartość
0 ⟷ 0	1	Stała określająca typ pakietu
1 ⟷ n		Łańcych określający zapytanie - nie może zawierać znaku \000
n+1 ⟷ n+1	\000	Znak końca string'a (\000) - a tym samym końca pakietu

5.4 StatementPackage

Otrzymanie od serwera ID wysłanego zapytania.

Od - do	Wartość	Zawartość
0 ⟷ 0	2	Stała określająca typ pakietu
1 ⟷ 4 (8)	unsigned long	Numer nadany utworzonemu zapytaniu

5.5 ParamStatementPackage

Wysłanie parametrów do już wysłanego zapytania na serwer.

Od - do	Wartość	Zawartość
0 ⟷ 0	3	Stała określająca typ pakietu
1 ⟷ 4 (8)	N - unsigned long	Całkowita długość tego pakietu
5 (9) ⟷ 8 (16)	unsigned long	Numer ID zapytania do którego przesyłamy parametry
9 (17) ⟷ 12 (24)	C - unsigned long	Liczba parametrów przekazywanych do zapytania
13 (25) ⟷ N		C wystąpień bloku opisującego pojedynczy parametr

5.5.1 Blok opisujący pojedynczy parametr

Pojedynczy parametr jest opisany następującą konstrukcją:

Od - do	Wartość	Zawartość
0 ⟷ 3 (7)	n (unsigned long)	Długość nazwy parametru (ze znakiem pustym)
4 (8) ⟷ n+2 (n+6)	unsigned long	Nazwa parametru
n+3 (n+7) ⟷ n+3 (n+7)	\000	Znak końca łańcucha
n+4 (n+8) ⟷ ...		Definicja wartości (patrz: Standardowy format struktury danych)

5.6 SimpleResultPackage

Odpowiedź od serwera z wynikami przetwarzania zapytania.

Od - do	Wartość	Zawartość
$0 \longleftrightarrow 0$	4	Stała określająca typ pakietu
$1 \longleftrightarrow \ldots$		Definicja wartości (patrz: Standardowy format struktury danych)

5.7 RemoteQueryPackage

Przetwarzanie rozproszone. Serwer prosi inny serwer o wykonanie podzapytania.

Od - do	Wartość	Zawartość
$0 \longleftrightarrow 0$	6	Stała określająca typ pakietu
$1 \longleftrightarrow 1$		Informacja o dereferencji (0-nie,1-tak)
$2 \longleftrightarrow N$		Remote LogicalID

5.8 RemoteResultPackage

Przetwarzanie rozproszone. Serwer odpowiada wynikami zapytania na zapytanie.

Od - do	Wartość	Zawartość
$0 \longleftrightarrow 0$	7	Stała określająca typ pakietu
$1 \longleftrightarrow \ldots$		Definicja wartości (patrz: Standardowy format struktury danych)

6 Struktura plików

Wszystkie pliki bezpośrednio związane z przesyłaniem danych przez sieć zawarte są w katalogu /TCPProto

6.1 Tcp.h i Tcp.cpp

Są to pliki zawierające „teoretycznie" niezależne funkcje do obsługi nawiązywania połączeń sieciowych i wysyłania nimi danych, a także serializowania prostych typów danych. Niestety istotna część tych funkcji jest nieoprawna (nie uważne stosowanie funkcji htonl - zamieniającej kolejność bajtów pomiędzy systemami opartymi o Big-endian i Little-endian).

Także użyte są nadal typu int, long - których rozmiar jest zależny od architektury na której kompilujemy system. Ponadto klasy te są użyte tylko w części przypadków - w pozostałych odczyty, konwersje i zapisy robione są ręcznie.

6.2 Package.h

Plik Package.h zawiera deklarację typu abstrakcyjnego z którego dziedziczą wszystkie klasy do przesyłania danych przez sieć:

```
class Package {
    public:
    enum packageType {
        RESERVED        = -1,   //to force nondefault deserialization method
        SIMPLEQUERY = 0,   //String
        PARAMQUERY = 1,    //String with $var
        STATEMENT = 2,     //Parser tree
        PARAMSTATEMENT = 3,//stmtNr + map
        SIMPLERESULT = 4,  //Result
        ERRORRESULT = 5,   //Parse/execute error package
        REMOTEQUERY = 6,   //remote reference
        REMOTERESULT = 7   //environment section
    };
    virtual packageType getType()=0;

    //returns error code, message buffer and size of the buffer
    //it doesn't destroy the buffer
    //first byte of the buffer is the resultType
    virtual int serialize(char** buffer, int* size)=0;

    //returns error code, gets buffer and it's size
    //it destroys the buffer
    virtual int deserialize(char* buffer, int size)=0;
    virtual ~Package(){}
};
```

Plik ten definiuje zatem identyfikatory pakietów zapisywane w pierwszym bajcie. Ponadto wymaga by klasa utożsamiana z pakietem posiadała metodę umożliwiającą wczytanie pakietu z danej tablicy bajtów (deserialize) i zapisanie pakietu do danej tablicy bajtów (serialize).

Ponadto w tym pliku nagłówkowym znajdują się deklarację wszystkich pakietów.

7 Stwierdzone wady protokołu

- Protokół ten nie będzie działał pomiędzy komputerami różniącymi się architekturą lub trybem kompilacji (big endian - little endian), (8, 16, 32, 64 bity).
- Brak określonych z góry rozmiarów pakietów i ich elementów - co utrudnia stronie odbierającej alokowanie buforów o odpowiednich rozmiarach.
- Brak możliwości przerwania zleconego zapytania w trakcie jego wykonania
- Każdy pakiet ma inną konstrukcję i sposób zapisywania treści.
- Brak negocjacji wstępnych przy nawiązywaniu połączenia (logowanie, porównanie wersji, synchronizacja czasu, strona kodowa znaków w łańcuchach)
- Generalny brak konsekwencji.

  Czasami kod korzysta z funkcji zawartych w pliku TCPIP.cpp,a innym razem realizuje całkowicie analogiczną operacje bezpośrednio.

  Dla przykładu - pola tekstowe są przesyłane na jeden z 3 sposobów:

  - Jako ciągi bajtów, poprzedzone liczbą świadczącą o ich długości (wliczając znak \000 na końcu).
  - Jako ciągi bajtów, poprzedzone liczbą świadczącą o ich długości (nie wliczając znak \000 na końcu).
  - Jako ciągi bajtów - z założeniem, że znak \000 kończy przesyłany napis.
- Brak organizacji wewnętrznej pakietów umożliwiającej wygodną rozbudowę protokołu (o nowe dane w poszczególnych pakietach)
- Obsługa maksymalnie maxint+1 obiektów (czyli np. 32768 na architekturach 16-bitowych).
- Protokół nie przewiduje możliwości przesyłania liczb całkowitych ujemnych.
- Duplikacja kodu - wiele fragmentów kodu jest skopiowanych (copy-paste) z innego miejsca. Można by tego uniknąć organizując kod w odpowiedni sposób.
- Istnieją trzy typy związane z przesyłaniem danych logicznych: Result::BOOLTRUE, Result::BOOLFALSE, Result::BOOL - każdy serializowalny z innym identyfikatorem. Prawdopodobnie któryś z nich powstał przez pomyłkę.