Projektowanie sieci typu Fieldbus

Zadanie projektowe nr 2: CRC ramki Modbus RTU

Szustkiewicz Krzysztof

Ivan Kaliankovich

Sprawozdanie

Spis treści

1) Wybrany algorytm

Na początku wybrany przez nas został algorytm iteracyjny CRC-16, ponieważ standardowy algorytm CRC-16 zgodny z protokołem MODBUS jest już dość efektywny i nie ma powszechnie stosowanych algorytmów, które byłyby znacznie szybsze przy zachowaniu dokładności. Obliczenie CRC może zostać zaimplementowane w sposób rekurencyjny, natomiast zwykle takie podejście jest mniej wydajne oraz bardziej skomplikowane od metody iteracyjnej.
W trakcie testowania zauważyliśmy jednak, że gdy chcemy obliczyć wiele sum CRC w krótkim czasie, to na większą wydajność pozwala algorytm tablicowy, dlatego postanowiliśmy pozostać przy **algorytmie tablicowym**.

2) Sposób pomiaru czasu

Do pomiaru czasu została użyta biblioteka *chrono* z poniższych powodów:

a) Wysoka precyzja: chrono dostarcza bardzo precyzyjne narzędzia pomiaru czasu, co pozwala na dokładne określenie czasu wykonania algorytmów nawet w mikrosekundach lub nanosekundach. Biblioteka ta oferuje znacznie większą precyzję od biblioteki *time.*
b) Kros-platformowość: Biblioteka chrono jest częścią standardu C++ i jest dostępna na wielu platformach, co oznacza, że możliwe jest korzystanie z tych samych metod pomiaru czasu niezależnie od systemu operacyjnego.
c) Uniwersalność: chrono oferuje różne API do różnych zegarów czasu, takie jak **system_clock(), steady_clock() i high_resolution_clock(),** które mogą być dostosowane do konkretnych potrzeb pomiaru czasu.
d) Prostota użycia: Dla pomiaru czasu wystarczy wywołać dwie funkcję: uruchomienia zegara oraz jego zatrzymania.
e) Zabezpieczenia przed różnicami czasu systemowego: chrono dostarcza narzędzia do kontrolowania różnic czasu systemowego, co jest ważne w przypadku precyzyjnych pomiarów czasu na wielu platformach jednocześnie.

3) Kod źródłowy

Poniżej przedstawiam kod inicjalizacji tablic oraz funkcji **getTableCRC()** liczącej sumę CRC metodą tablicową. Cały kod źródłowy można znaleźć tu: github

```
// Wielomian generujący dla CRC-16 (Reversed polynomial: 0xA001)
const uint16_t crc16_polynomial = 0xA001;

// Tablice LowByte i HiByte CRC-16
uint8_t crc16_table_low[256];
uint8_t crc16_table_high[256];

// Inicjalizacja tablic CRC-16
void initializeCRCTables() {
    for (int i = 0; i < 256; i++) {
        uint16_t crc = i;
        for (int j = 0; j < 8; j++) {
            if (crc & 1) {
                crc = (crc >> 1) ^ crc16_polynomial;
            } else {
                crc >>= 1;
            }
        }
        crc16_table_low[i] = static_cast<uint8_t>(crc & 0xFF);
        crc16_table_high[i] = static_cast<uint8_t>((crc >> 8) & 0xFF);
    }
}

// Funkcja do obliczania CRC-16
uint16_t getTableCRC(const std::vector<uint8_t>& data) {
    uint8_t crc_low = 0xFF;
    uint8_t crc_high = 0xFF;
    for (uint8_t byte : data) {
        uint8_t index = crc_low ^ byte;
        crc_low = crc16_table_low[index] ^ crc_high;
        crc_high = crc16_table_high[index];
    }

    return (static_cast<uint16_t>(crc_high) << 8) |
static_cast<uint16_t>(crc_low);
}
```

4) Przykłady wyniku

Przykład wyniku obliczenia CRC dla sekwencji: efefefefe1231323323198 przy n = 1490323 powtórzeń algorytmu.

a) Metoda iteracyjna:

```
C:\Fieldbus\proj2>Modbus
Podaj sekwencje bajtow w notacji hex: efefefefe1231323323198
Podaj liczbe n powtorzen algorytmu CRC (1 do 1000000000): 1490323
Suma CRC po 1490323 powtorzeniach: 0xcd47
Czas: 1.17274s
```

1.17274s

b) Metoda tablicowa przy identycznej sekwencji oraz ilości powtórzeń:

```
C:\Fieldbus\proj2>Modbus
Podaj sekwencje bajtow w notacji hex: efefefefe1231323323198
Podaj liczbe n powtorzen algorytmu CRC (1 do 1000000000): 1490323
Suma CRC po 1490323 powtorzeniach: 0x47cd
Czas: 0.488238s
```

5) Wnioski

Metoda tablicowa pozwala na znaczące przyśpieszenie procesu w przypadku, gdy jest dużo powtórzeń. W sytuacji, gdzie nie ma dużo sum CRC do obliczenia algorytm iteracyjny jest preferowany, ponieważ jest bardzo prosty, nie wymaga dużo pamięci i jest wystarczająco szybki dla większości przypadków.