Technologie sieciowe 3

Gabriel Wechta

7.05.2020

1 Zadanie 1

W pierwszym zadaniu mamy przeczytać ciąg bitów z pliku, ramkując je według zasady rozpychania bitów, dodać kod umożliwiający sprawdzenie poprawności za pomocą CRC, po czym na początku i na końcu tak przetworzonego ciągu, mamy dodać flagi początku i końca wiadomości.
Zadanie zrealizuję w c++ symulując przesłanie danych poprzez zapis do pliku zakodowanych ciągów, ponowne odczytanie ich i odkodowanie. Program wyświetli oryginalne dane oraz odkodowane.

1.1 Opis działania programu

Zważywszy na łatwość w operacjach obcinania i iterowania po stringach, ich użyję do reprezentacji ciągów bitów.
Program czyta pierwszą linijkę pliku Z.txt, która zawiera ciąg '0' i '1', następnie dzieli ją na fragmenty o stałej długości $PACKAGESIZE$.

```
if (input_file.is_open()) {
    getline(input_file, line);
    for (int i = 0; i <= line.length() / PACKAGE_SIZE; i++) {
        package = line.substr(i * PACKAGE_SIZE, PACKAGE_SIZE);
        if (package.empty()) break;
        package = fiveOnesFix(package);
        package = package + getCRC(package);
        package = begEndWrap(package);
        output_file << package << "\n";
    }
}
```

Biorąc pod uwagę, że wysłana ramka ma zawierać ciąg otwierający i zamykający, takie rozwiązanie wydaje się być najlepsze. Nie tracimy miejsca na dopychanie ramki, co miałoby miejsce, gdybyśmy chcieli ramki o stałej długości. Zabezpieczenie polegające na dodaniu '0' po pięciu '1', może rozszerzyć wiadomość o $PACKAGESIZE\ div\ 5$, co przy payloadzie długości 16 (jak u mnie) może rozszerzyć każdą ramkę o 3 bity. Lepiej jest ustalić długość payloadu, niż ramki.

Z drugiej strony zmienna długość może być nieakceptowalna przez różne technologie przesyłu sygnału. Natomiast, jako że zadanie jest tylko symulacją, zastosowałem zmienną długość ramki.
Program iteruje po stringu, szukając serii pięciu '1', na których końcu umieszcza '0'. Do wygenerowania 3-bitowego CRC używam dizelnika — "1011", ponadto zaimplementowałem prosty XOR na stringach. 3-bitowy CRC powinien być wystarczająco bezpieczny dla danych długości 16–19 bitów. Wygenerowany CRC umieszczany jest na końcu stringa. Po czym dodawana jest flaga otwierającą i zamykająca. Poniżej ciała funckji realizujących powyższe procedury.

Listing 1: "Dodawanie '0' po pięciu '1'"

```
string fiveOnesFix(string line) {
    int ones_counter = 0;
    for (int i = 0; i < line.length(); i++) {
        if (line[i] == '1') {
            ones_counter++;
            if (ones_counter == 5) {
                line.insert(i + 1, "0");
                i++;
                ones_counter = 0;
            }
        } else {
            ones_counter = 0;
        }
    }
    return line;
}
```

Listing 2: "Dodawanie CRC"

```
string getCRC(string line) {
    string divisor = "1011";
    line = line + "000";
    for (int i = 0; i <= line.length() - 4; i++) {
        if (line[i] == '1') {
            for (int j = 0; j < 4; j++) {
                if (line[i + j] == divisor[j]) {
                    line[i + j] = '0';
                } else {
                    line[i + j] = '1';
                }
            }
        }
    }
    return line.substr(line.length() - 3, line.length() - 1);
}
```

Listing 3: "Dodawanie flagi otwierającej i zamykającej"

```
string begEndWrap(string line) {
    return "01111110" + line + "01111110";
}
```

Następnie ciąg zapisywany jest w nowej linijce w W.txt.
Do odkodowania, powyższe operacje wykonywane są w odwrotnej kolejności.

Listing 4: "Dekodowanie"

```
string decode(string line) {
    if (line.substr(0, 8) != "01111110") return "broken";
    else line = line.substr(8, line.length());

    if (line.substr(line.length() - 8, line.length()) != "01111110")
        return "broken";
    else line = line.substr(0, line.length() - 8);

    if (checkCRC(line)) {
        line = line.substr(0, line.length() - 3);
    } else return "broken";

    int ones_counter = 0;
    for (int i = 0; i < line.length(); i++) {
        if (line[i] == '1') {
            ones_counter++;
            if (ones_counter == 5) {
                line.erase(i + 1, 1);
                ones_counter = 0;
            }
        } else {
            ones_counter = 0;
        }
    }
    return line;
}
```

Przy czym w przypadku gdy procedura sprawdzająca CRC stwierdzi uszkodzenie ramki, string zastępowany jest komunikatem "broken". Poniżej ciało funkcji sprawdzającej CRC.

Listing 5: "Sprawdzanie CRC"

```
/** line is already appended by crc code */
bool checkCRC(string line) {
    string divisor = "1011";
    for (int i = 0; i <= line.length() - 4; i++) {
        if (line[i] == '1') {
            for (int j = 0; j < 4; j++) {
                if (line[i + j] == divisor[j]) {
                    line[i + j] = '0';
                } else {
                    line[i + j] = '1';
                }
            }
        }
    }
    for (char i : line) {
        if (i == '1') return false;
    }

    return true;
}
```

Jeżeli wszystko się zgadza program wypisuje ciąg.
Pełen kod źródłowy dołączony jest do sprawozdania, w pliku zadanie1.cpp.

2 Zadanie 2

Treść zadania 2 pozostawia wiele pola do interpretacji, nie wiem, czy dziury w mojej symulacji uda mi się upchać słownym wyjaśnieniem i machaniem rękami, ale z uporem godnym lepszej sprawy spróbuję.

W zadaniu do reprezentacji łącza mamy użyć tablicy, a propagacje sygnału ma symulować propagacja wartości do sąsiednich komórek tablicy. Naturalnym kandydatem do programu okazuje się ponownie być c++, z uwagi na wrodzoną umiejętność pętli for do zgrabnego inkrementowania i dekrementowania indeksów.

Próbując jak najbardziej zasymulować rzeczywistą sytuację, postanowiłem reprezentować nadajnik poprzez obiekt klasy ze statycznym (współdzielonym) polem będącym tablicą typu int. Przez bardzo krótki czas rozważałem użycie wątków, natomiast, na szczęście, szybko ten pomysł umarł. Pomimo, że wartości tablicy są zmieniane sekwencyjnie, cała symulacja traktuje je jak operacje równoległe. Poniżej ciało klasy Feeder, będącą reprezentacją nadajnika.

Jeszcze parę słów od CSMA/CD, zanim przejdę do prezentacji rozwiązania. Protokół nie jest już raczej stosowany, zważywszy na to, że problem, który rozwiązuje, istnieje tylko w połączeniach half-duplex, których teraz się nie używa. W CSMA/CD nadajnik orientuje się o uszkodzeniu pakietu, bazując na zmianie w natężeniu (fizycznym) sygnału. U mnie kolizja jest symulowana poprzez zmianę wartości przesyłanego sygnału na int(3). Protokół zakłada, że węzły są nadajnikami prądu o stabilizowanym natężeniu, wtedy sygnał informujący o odebraniu uszkodzonej ramki (JAM) jest wyższy od normalnie nadawanych wartości, u mnie int(4).

Poniżej ciało klasy Feeder, będącej reprezentacją nadajnika.

Listing 6: "Klasa Feeder"

```
class Feeder {
private:
    int location = 0;
    int signal_value = 0;
    int signal_location = 0;


public:
    static int canal[10];

    explicit Feeder(int location, int id) {
        this->location = location;
        this->signal_value = id;
        this->signal_location = location;
    }

    static void showCanal() {
        cout << "{ ";
        for (int i : Feeder::canal) {
            cout << i << ", ";
        }
        cout << "}" << endl;
    }

    bool propagateRight(Feeder target) {
        if (this->signal_location == target.getLocation()) {
            cout << "Right target reached. ";
            if (Feeder::canal[target.getLocation()] == 3) {
                cout << "Jam detected. ";
            } else cout << "Everything good. ";
            cout << endl;
            return true;
        }
```

```
    this->signal_location++;
    if (Feeder::canal[this->signal_location] != 0) {
        Feeder::canal[this->signal_location - 1] = 3;
        Feeder::canal[this->signal_location]
            = Feeder::canal[this->signal_location - 1];
        return false;
    }
    if (Feeder::canal[this->signal_location - 1] == 0) {
        Feeder::canal[this->signal_location - 1] = this->signal_value;
    }
    Feeder::canal[this->signal_location]
        = Feeder::canal[this->signal_location - 1];
    Feeder::canal[this->signal_location - 1] = 0;
    return false;
}

bool propagateLeft(Feeder target) {
    if (this->signal_location == target.getLocation()) {
        cout << "Left target reached. ";
        if (Feeder::canal[target.getLocation()] == 3) {
            cout << "Jam detected. ";
        } else cout << "Everything good. ";
        cout << endl;
        return true;
    }
    this->signal_location--;
    if (Feeder::canal[this->signal_location] != 0) {
        Feeder::canal[this->signal_location + 1] = 3;
        Feeder::canal[this->signal_location]
            = Feeder::canal[this->signal_location + 1];
        return false;
    }
    if (Feeder::canal[this->signal_location + 1] == 0) {
        Feeder::canal[this->signal_location + 1] = this->signal_value;
    }
    Feeder::canal[this->signal_location]
        = Feeder::canal[this->signal_location + 1];
    Feeder::canal[this->signal_location + 1] = 0;
    return false;
}

int getLocation() {
    return this->location;
}

void signalJam() {
```

```
        int l, r;
        int canal_size = sizeof(Feeder::canal) / sizeof(int);
        for (l = this->location, r = this->location;; l--, r++) {
            if (l < 0 && r > canal_size) break;
            if (l < 0) {
                l++;
                Feeder::canal[l] = 0;
            } else {
                Feeder::canal[l] = 4;
                Feeder::canal[l + 1] = 0;
            }

            if (r > canal_size) {
                r--;
                Feeder::canal[r] = 0;
            } else {
                Feeder::canal[r] = 4;
                Feeder::canal[r - 1] = 0;
            }
            Feeder::showCanal();
        }
        cout << "Jam signal was populated. ";
    }
};
```

Opis metod i konstruktora:

1. Feeder(int location, int id) - konstruktor. Pole location ustala miejce "podłączenia" obiektu do łącza, id jest wartością nadawana przez dany obiekt. Zakazane wartości to 0, 3 i 4.

2. static void showCanal() - metoda drukuje aktualny stan łącza.

3. bool propagateRight(Feeder target) - po rozpoczęciu propagacji, sygnałem dalej zajmuje się obiekt klasy który ją wysłał. Metoda propagateRight przesuwa o jeden w prawo wartość nadaną przez obiekt, który ją wywołał. Jeżeli natrafi na niepusty fragment łącza (tym samym uszkadzajac sygnał) koduje uszkodzone dane jako int(3), propaguje je aż trafi do celu.

4. bool propagateLeft(Feeder target) - analogicznie.

5. void signalJam() - metoda, po której wywołaniu w obu kierunkach łącza wysyłany jest sygnał int(4) kodujący JAM.

2.1 Przykład 1 - poprawne przesłanie danych

Listing 7: "Poprawne przesłanie sygnału"

```
Feeder comp1(1, 1);
Feeder comp2(8, 2);

/* one-way signal scenario */
while (!comp2.propagateLeft(comp1)) {
    Feeder::showCanal();
}
```

Output:

```
{ 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, }
{ 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, }
{ 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, }
{ 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, }
{ 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, }
{ 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, }
{ 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, }
Left target reached. Everything good.
```

2.2 Przykład 2 - Kolizja oraz nadawanie JAM

Listing 8: "Kolizja oraz nadawanie JAM"

```
/* simultaneous signal scenario */
while (!comp1.propagateRight(comp2) && !comp2.propagateLeft(comp1)) {
    Feeder::showCanal();
}
comp2.signalJam();
```

Output:

```
{ 0, 0, 1, 0, 0, 0, 0, 2, 0, 0, }
{ 0, 0, 0, 1, 0, 0, 2, 0, 0, 0, }
{ 0, 0, 0, 0, 1, 2, 0, 0, 0, 0, }
{ 0, 0, 0, 0, 3, 3, 0, 0, 0, 0, }
{ 0, 0, 0, 3, 0, 0, 3, 0, 0, 0, }
{ 0, 0, 3, 0, 0, 0, 0, 3, 0, 0, }
{ 0, 3, 0, 0, 0, 0, 0, 0, 3, 0, }
Right target reached. Jam detected.
{ 0, 3, 0, 0, 0, 0, 0, 0, 4, 0, }
{ 0, 3, 0, 0, 0, 0, 0, 4, 0, 4, }
{ 0, 3, 0, 0, 0, 0, 4, 0, 0, 0, }
{ 0, 3, 0, 0, 0, 4, 0, 0, 0, 0, }
{ 0, 3, 0, 0, 4, 0, 0, 0, 0, 0, }
{ 0, 3, 0, 4, 0, 0, 0, 0, 0, 0, }
{ 0, 3, 4, 0, 0, 0, 0, 0, 0, 0, }
{ 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, }
{ 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, }
Jam signal was populated.
```

2.3 Uwagi

Model przeze mnie zaprezentowany (klasa ze statycznym polem) moze posłużyć do przeprowadzenia pozostałych sytuacji, w których wykorzystuje się CSMA-/CD.
Plik z kodem źródłowym, zadanie2.cpp, dołączam do sprawozdania.