Karol Szymończyk 243434 15.04.2019r. Wrocław

Lista 3

♦ Zadanie 1

Celem zadania pierwszego było napisanie programu, który dokonuje ramkowania zgodnie z zasadą ”rozpychania bitów” oraz weryfikujący poprawność ramki metodą CRC. Program potrafi także zrealizować procedurę odwrotną, tzn. odczytać plik wynikowy i dla poprawnych danych CRC przepisuje jego zawartość tak, aby otrzymać kopię oryginalnego pliku źródłowego.

- **Ramka[1] –** (pot. Pakiet) sformatowana jednostka informacji przesyłana poprzez sieć. Ramka składa się z pewnego rodzaju ”opakowania” oraz obszaru danych, w którym znajdują się konkretne dane, które mają zostać przesłane za pomocą danego pakietu. Wszystkie dane w określonej sesji, które mają być przesłane poprzez sieć, dzielone są na serię pakietów. W niektórych rodzajach sieci pakiety składające się na sesję nie muszą być przesyłane tą samą trasą.

- **Zasada rozpychania bitów –** jej zadaniem jest uniknięcie niepoprawnego odczytania wiadomości poprzez wcześniejsze zakończenie czytania po natrafieniu na sekwencję kończącą ramkę, która znajdowała się w ciągu danych umieszczonych w obszarze danych. Polegała ona na ”rozepchaniu” sekwencji bitów 111111 podobnej do sekwencji kończącej oraz rozpoczynającej 01111110. Rozepchanie to polegało na umieszczenie „0” po ciągu pięciu „1”. Przykładowo wiadomość 1111111 po rozepchaniu wyglądała następująco: 11111011. W przypadku wiadomości 111110 zostaje ona ”rozepchana” do następującej postaci: 1111100.

- **Metoda CRC[2] –** (*Cyclic Redundancy Check)* czyli cykliczny kod nadmiarowy. Jest to system sum kontrolnych wykorzystywany do wykrywania przypadkowych błędów pojawiających się podczas przesyłania danych binarnych. N-bitowy cykliczny kod nadmiarowy definiuje się jako resztę z dzielenia ciągu danych przez (n+1)-bitowy ustalany dzielnik CRC.

[1] https://pl.wikipedia.org/wiki/Pakiet_telekomunikacyjny
[2] https://pl.wikipedia.org/wiki/Cykliczny_kod_nadmiarowy

Napisany przeze mnie program ramkuje w następujący sposób:
Odczytuje sekwencję podaną w pliku ‘z.txt’, później wylicza CRC wiadomości, korzystając z biblioteki CRC32 co oznacza, że sekwencja CRC jest 32-bitowa. W przypadku gdy CRC jest krótsze niż 32 symbole, dodaję odpowiednią liczbę zer na jego początek. Następnie ciąg wiadomości oraz jej CRC poddawany jest ”rozpychaniu bitowemu”. Odbywa się to w pętli wyszukującej ciągu pięciu „1”, a następnie w przypadku jego znalezienia dodane zostaje „0” po jego wystąpieniu. Na koniec do otrzymanego ciągu zer oraz jedynek dodawane są sekwencje rozpoczynające oraz kończące (0111110) oraz tak otrzymana ramka zostaje zapisana do pliku ‘w.txt’.

Fragment kodu odpowiadający za ramkowanie.

```
class Encode extends Code{

  private final static String signCode = "01111110";

  static void encode(String readFile, String writeFile) throws IOException {
    String message = readFromFile(readFile);
    String crc = calcCRC(message);
    String messageWithCRCR = message + crc;
    messageWithCRCR = stretch(messageWithCRCR);
    PrintWriter pw = new PrintWriter(writeFile);
    pw.println(signCode + messageWithCRCR + signCode);
    pw.close();
  }

  private static String stretch(String message) {
    StringBuilder str = new StringBuilder(message);
    int counter = 0;
    for (int i = 0; i < str.length(); i++) {
      Character c = str.charAt(i);
      if (c.equals('1') && counter < 5) {
        counter++;
        if (counter == 5) {
          str.insert( offset: i + 1,  c: '0');
          counter = 0;
        }
      } else {
        counter = 0;
      }
    }
    return str.toString();
  }
}
```

Fragment kodu, z którego metod korzystamy w powyższym fragmencie.

```
abstract class Code {

  static String readFromFile(String fileName) throws IOException {
    return new String(Files.readAllBytes(Paths.get(fileName)));
  }

  static String calcCRC(String message) {
    CRC32 checksum = new CRC32();
    checksum.update(message.getBytes(),  off: 0, message.getBytes().length);
    return make32(Long.toBinaryString(checksum.getValue()));
  }

  private static String make32(String crc) {
    StringBuilder str = new StringBuilder(crc);
    while (str.length() < 31) {
      str.insert( offset: 0,  c: '0');
    }
    return str.toString();
  }
}
```

Program odczytuje wiadomość z ramki w następujący sposób:
Na początku zostaje wczytana cała ramka z pliku ‘w.txt’. Później zostają znalezione oraz usunięte sekwencje rozpoczynające oraz kończące oraz wyodrębniona zostaje rozepchana bitowo wiadomość wraz z jej CRC. Następnie usuwamy dodatkowe „0” powstałe w wyniku rozpychania poprzez odszukanie ciągów pięciu „1” oraz usunięcia „0” występującego po znalezionej sekwencji. Teraz rozdzielona zostaje wiadomość oraz CRC poprzez usunięcie ostatnich 32 znaków (32-bitowe CRC). Przystępujemy do sprawdzenia czy podczas „przesyłania” zostało wtrącone przekłamanie. Aby tego dokonać obliczamy CRC wyodrębnionej wiadomości oraz porównujemy je z CRC przesłanym w wiadomości. Jeżeli zostanie wykryta nieprawidłowość, w pliku ‘zc.txt’ pojawi się wiadomość „ Distortion found! Please try again.”. W przypadku prawidłowego wyniku porównania do tego samego pliku zostanie zapisana wiadomość pozbawiona ramki.

Aby uzyskać przekłamanie po pierwszym wywołaniu programu w klasie *Main.java* należy zakomentować część odpowiadającą za zakodowywanie. Następnie w pliku ‘w.txt’ znajdującym się w folderze *files* należy dokonać przekłamania oraz uruchomić program.

Fragment klasy Main.java, w którym należy zakomentować linijkę Encode w celu uzyskania przekłamania.

```
Encode.encode( readFile: dest_dir + z,  writeFile: dest_dir + w);

Decode.decode( readFile: dest_dir + w,  writeFile: dest_dir + zc);
```

Fragment klasy Decode odpowiedzialniej za odczytanie wiadomości z ramki.

```
class Decode extends Code {

  static void decode(String readFile, String writeFile) throws IOException {
    String code = readFromFile(readFile);
    String messageWithCRC = removeSignCode(code);
    messageWithCRC = unstretch(messageWithCRC);
    String message = removeCRC(messageWithCRC);
    String crc = removeMessage(messageWithCRC);
    PrintWriter pw = new PrintWriter(writeFile);
    if (!checkCrc(crc, message)) {
      System.out.println("Distortion found! Please try again.");
      pw.println("Distortion found! Please try again.");
      pw.close();
      return;
    }
    pw.println(message);
    pw.close();
  }

  private static boolean checkCrc(String crc, String message) {
    String currCrc = calcCRC(message);
    return currCrc.equals(crc);
  }

  private static String unstretch(String message) {
    StringBuilder str = new StringBuilder(message);
    int counter = 0;
    for (int i = 0; i < str.length(); i++) {
      Character c = str.charAt(i);
      if (c.equals('1') && counter < 5) {
        counter++;
        if (counter == 5) {
          str.deleteCharAt(i + 1);
          counter = 0;
        }
      } else {
        counter = 0;
      }
    }
    return str.toString();
  }
```

Przykład 1

Plik 'z.txt'

```
101000101100110000110100111000111
```

Opisany plik 'w.txt'

```
01111110101000101100110000110100111000111101011100001011110100111000110101111110
|       |                                |                                |      |
  kod              wiadomość                           CRC                    kod
```

Plik 'zc.txt'

```
101000101100110000110100111000111
```

Przykład 2

Plik 'z.txt'

```
1111111
```

Prawidłowy opisany plik 'w.txt'

```
01111110111111011011101010100010001000110100110101111110
|       |        |                                |      |
  kod   wiadomość              CRC                    kod
```

Wprowadzone przekłamanie do pliku 'w.txt'

```
01111110111010110111010101000100010001101001101011111110
|       |    |    |                                |      |
  kod   wiadomość              CRC                    kod
             |
        przekłamanie
```

Wynik w pliku 'zc.txt'

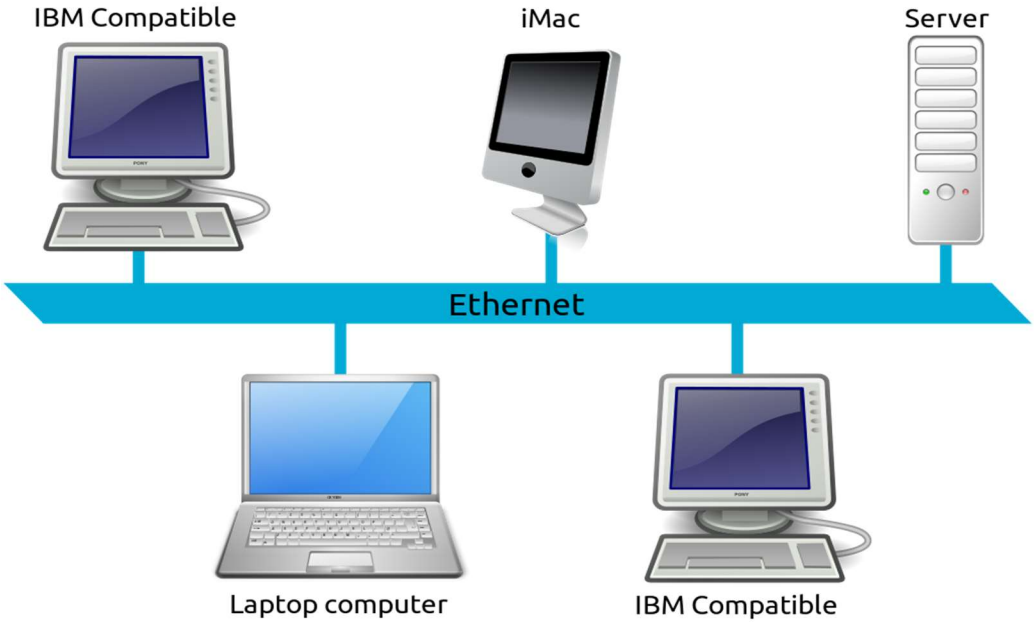
```
Distortion found! Please try again.
```

♦ Zadanie 2

Celem zadania drugiego było napisanie programu, który służy do symulowania ethernetowej metody dostępu do medium transmisyjnego (CSMA/CD).

- **Ethernet**[3] **–** technika, w której zawarte są standardy wykorzystywane w budowie głównie lokalnych sieci komputerowych. Obejmuje ona specyfikację przewodów oraz przesyłanych nimi sygnałów.

Schemat sieci Ethernet.



- **CSMA/CD**[4] **–** (*Carrier Sense Multiple Access / with Collision Detection*) protokół wielodostępu CSMA z badaniem stanu kanału oraz wykrywaniem kolizji. Wielodostęp polega na tym, że każde urządzenie lub węzeł w sieci, które chce przesłać dane, nasłuchuje łącza, sprawdzając czy jakieś inne urządzenie nie przesyła danych w linii transmisyjnej. Dane będą wysłane jedynie wtedy, gdy nie zostanie wykryty żaden sygnał świadczący o tym, że jakieś urządzenie w sieci wysyła dane. Istnieje możliwość, że dwa lub więcej urządzeń przystąpi do wysyłania danych w tej samej chwili lub zanim sygnał z pierwszego węzła dotrze do drugiego. W takiej sytuacji żadne z nich nie wykryje sygnału nośnego drugiego. W efekcie obydwa urządzenia wysyłając dane w tym samym czasie spowodują kolizję w sieci Ethernet. Możliwość wystąpienia takiej sytuacji rodzi potrzebę stworzenia mechanizmów pozwalających tę kolizję wykryć i wyeliminować jej skutki.

[3] https://pl.wikipedia.org/wiki/Ethernet
[4] https://pl.wikipedia.org/wiki/CSMA/CD

Napisany przeze mnie program posiada parametry, za pomocą których możemy dostosować naszą „sieć Ethernet” do potrzeb testów. Możemy ustawić długość naszej sieci, długość interwałów oraz maksymalny czas po którym stację będą chciały wysłać sygnał.

Fragment kodu odpowiadający za ustawienie parametrów symulacji.

```
const EthernetLength = 30
const Interval = Millisecond * 300
const MaxWait = 50
```

Sieć Ethernet zaimplementowałem jako „serwer”, w którym znajduję się tablica z poszczególnymi polami, na początku wypełniona zerami. Serwer ten posiada możliwości *write*, która pozwala na zapisanie podanej wartości na podanej pozycji oraz *read* pozwalająca odczytać wartość na podanym miejscu. Dzięki temu rozwiązaniu wątki symulujące stacje podłączone do nasz sieci nie będą w jednym czasie zmieniać jej wartości.

Fragment kodu „serwera” symulującego sieć Ethernet.

```
func ethernet(reads chan *readOp, writes chan *writeOp) {
    var state []string
    state = fillSlice(state, "0")
    printConnections()
    printEthernet(state)
    for {
        select {
        case write := <-writes:
            isRight := true
            switch write.val {
            case "0":
                if state[write.key] == write.sign || state[write.key] == "=" {
                    state[write.key] = write.val
                }
            default:
                if state[write.key] == "0" || state[write.key] == write.val {
                    state[write.key] = write.val
                } else {
                    state[write.key] = "="
                    isRight = false
                }
            }
            printEthernet(state)
            write.resp <- isRight

        case read := <-reads:
            read.resp <- state[read.key]
        }
    }
}
```

Stacje zaimplementowałem jako osobne wątki, które posiadają swoje miejsce podłączenia do tablicy imitującej sieć oraz swój sygnał, który będą nadawać.

Fragment kodu odpowiadający za implementację stacji.

```
func station(signal string, place int, reads chan *readOp, writes chan *writeOp) {
    stations[place] = signal
    connections[place] = "|"
    wait := getRandomTime()
    cu := make(chan bool, 1)
    cd := make(chan bool, 1)
    counter := 0
    Sleep(wait)
    for {
        if isFree(place, reads) {
            sendSignal(place, signal, signal, writes)
            Sleep(Interval)
            go propagateUp(place, signal, signal, writes, cu)
            go propagateDown(place, signal, signal, writes, cd)
            Sleep(Interval)

            c1 := <-cu
            c2 := <-cd

            sendSignal(place,  val: "0", signal, writes)
            Sleep(Interval)

            go propagateUp(place,  signal: "0", signal, writes,  cu: nil)
            go propagateDown(place,  signal: "0", signal, writes,  cd: nil)

            if !c1 || !c2 {
                counter++
                wait = getTimeToWait(counter)
            } else {
                counter = 0
                wait = getRandomTime()
            }
            Sleep(wait)
        } else {
            Sleep(Interval)
        }
    }
}
```

Fragment kodu uruchamiającego stacje.

```
go station( signal: "A",  place: 10, reads, writes)
go station( signal: "B",  place: 2, reads, writes)
```

Stacje te nadają swoje sygnały oraz mają możliwość wykrycia kolizji, gdy do niej dojdzie. Wtedy następuje procedura rozwiązania konfliktu. Polega ona na tym że stacje, między którymi doszło do konfliktu zaczynają losować przerwy między następną próbą wysłania komunikatu kolejno z przedziałów 2^0, 2^1, 2^2, …,2^{10} dla 1, 2, 3, …,10 konfliktów pod rząd. Następnie po 10 nieudanych próbach rozwiązania konfliktów, procedura jest powtarzana 6 razy dla przedziału 2^{10}. Jeżeli konflikt nadal zostanie nierozwiązany oznacza to że sieć ma źle dobrane parametry i nie będzie działała poprawnie.

Fragment kodu odpowiedzialny za losowanie liczby interwałów.

```
func getTimeToWait(counter int) Duration {
    source := rand.NewSource(Now().UnixNano())
    generator := rand.New(source)
    var MaxWait = 0
    switch {
    case counter < 10:
        MaxWait = int(math.Pow( x: 2, float64(counter)))
    case counter < 17:
        MaxWait = int(math.Pow( x: 2,  y: 10))
    default:
        {
            println( args...: "Ethernet is not working correctly")
            os.Exit( code: 1)
        }
    }
    r := Duration(generator.Intn(MaxWait))
    return r * Interval
}
```

Przykładowe wywołania programu :

Pusta symulacja sieci Ethernet z podłączonymi dwiema stacjami.

```
[    B               A                                                  ]
[    |               |                                                  ]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Propagacja sygnału ze stacji B w symulacji sieci.

```
[    B               A                                                  ]
[    |               |                                                  ]
[B B B B B B B B B B B B B B B B B B B B B B B B 0 0 0 0 0 0 0 0 0 0 0 0]
```

Przykład konfliktu między stacjami A i B

```
[    B               A                                                  ]
[    |               |                                                  ]
[B B B = = = = = = = = A A A A A A A 0 B B B B B B B B B B B B B B B B]
```

♦ Wnioski

Z zadania pierwszego możemy nauczyć się jak pakowane są informacje podczas przesyłania oraz jak wygląda kontrola ich poprawności. Możemy wywnioskować także jak skuteczna jest metoda kontroli CRC oraz jaką ma przewagę nad sprawdzaniem parzystości bitów. Przy sprawdzaniu parzystości prawdopodobieństwo niewykrycia przekłamania jest znacznie większe niż przy użyciu CRC.

Natomiast z zadania drugiego możemy dowiedzieć się jak działa sieć Ethernet oraz jak ważne jest odpowiednie dobranie parametrów. Podczas przeprowadzania wielu prób mogę wnioskować, że do prawidłowego działania tego typu sieci musimy dobrać odpowiednią długość tablicy do liczby stacji, próbujących się dzięki niej komunikować. Przy zbyt dużej liczbie stacji dochodzi do konfliktów zbyt często i część stacji zostaje zablokowanych.