Technologie sieciowe, sprawozdanie z zajęć laboratoryjnych z listy 3

Autor: Piotr Klepczyk 236408

1. Cel

Celem zadania było stworzenie dwóch programów, pierwszy miał przeprowadzać ramkowanie oraz sprawdzić poprawność ramkowania metodą CRC. Ramkowaniu ma być podany wejściowy plik tekstowy zawierający ciąg znaków składający się z ‘0’ i ‘1’ i zapisywać odpowiednio sformułowany ciąg do pliku tekstowego. Program ten powinien też realizować procedurę odwrotną to znaczy na wejściu otrzymać plik po ramkowaniu i go odtworzyć. Drugi program ma służyć do symulowania ethernetowej metody dostępu do medium transmisyjnego, gdzie wspólne łącze będzie realizowane przy pomocy tablicy, a propagacja sygnału jest realizowana jako propagacja wartości do sąsiednich komórek tablicy.

2. Realizacja

2.1.Ramkowanie

2.1.1. Opis programu

Program został napisany w języku Java. Program jest w stanie odczytać dane wejściowe z pliku oraz zapisać dane wyjściowe do pliku przy zadanym kluczu CRC. Odczyt z pliku został stworzony w oparciu o bibliotekę `Scanner`, funkcja za to odpowiadająca wygląda następująco:

```
1     public String getFile(String name) {
2           String t;
3           File file = new File(name);
4           Scanner scan = null;
5           try {
6                 scan = new Scanner(file);
7           } catch(FileNotFoundException e) {
8                 e.printStackTrace();
9           }
10          t = scan.nextLine();
11          return t;
12    }
```

Funkcja „getFile” programu „Framing”

Jak widać program na wyjściu zwraca zawartość odczytaną z pliku, którego nazwa jest podawana na wejściu do zmiennej `name`, do zmiennej. By funkcja działała poprawnie plik tekstowy który odczytuje musi być umiejscowiony wraz z plikami programu. Zakładamy również że w pliku znajduje się nieprzerwany ciąg znaków.

Kolejną funkcjonalnością programu jest tworzenie pliku tekstowego i zapisywanie do niego treści, funkcja odpowiadająca za to wygląda następująco:

```
1     public void createFile(String s,String name) {
2           PrintWriter writer = null;
3           try {
4                 writer = new PrintWriter(name);
5           } catch (FileNotFoundException e) {
6                 e.printStackTrace();
7           }
8           writer.print(s);
9           writer.close();
10    }
```

Funkcja „createFile” programu „Framing”

Funkcja ta na wejściu otrzymuje dwie wartości, jest to tekst który zostanie zapisany i nazwa pliku do którego zostanie on napisany (odpowiednio zmienne `s` i `name`). Funkcja ta korzysta z biblioteki `PrintWriter` która odpowiada za zapis zmiennej typu String do pliku. Funkcja ta może nadpisać istniejący już plik lub stworzyć nowy. Utworzony plik znajduje się wraz z plikami programu.

Do sprawdzenia poprawności ramkowania jest wykorzystywana klasa `Crc` która zawiera metody pozwalające wyliczyć kod CRC przy zadanym kluczu oraz sprawdzić poprawność uzyskanego ciągu. Metoda CRC polega na wyznaczeniu kodu CRC dla zadanego ciągu bitów („0” i „1”) i dopisanie go na jego końcu, dla danego klucza. Długość kodu CRC jest zawsze o jeden mniejsza niż długość klucza którego minimalna długość to dwa. Opisowo ten algorytm, dla klucza długości n, wygląda następująco:

```
1. dodajemy do ciągu danych n-1 wyzerowane bity,
2. w linii poniżej wpisujemy n-bitowy dzielnik CRC,
3. jeżeli mamy 0 nad najstarszą pozycją dzielnika, to przesuwamy dzielnik w prawo o jedną
   pozycję, aż do napotkania 1,
4. wykonujemy operację XOR pomiędzy bitami dzielnika i odpowiednimi bitami ciągu danych,
   uwzględniając dopisane n-1 bitów
5. wynik zapisujemy w nowej linii poniżej,
6. jeżeli liczba bitów danych jest większa lub równa n, przechodzimy do kroku 2,
7. n-1 najmłodszych bitów stanowi szukany kod CRC
```

Opis algorytmu wyznaczania kodu CRC

W wyniku przeprowadzenia tego algorytmu w miejscu dodanych wyzerowanych bitów otrzymamy kod CRC, który należy dopisać na końcu naszego ciągu, po przeprowadzeniu tego algorytmu dla otrzymanego w ten sposób ciągu z tym samym kluczem otrzymamy ciąg samych zer, w ten sposób można sprawdzić czy z naszym ciągiem nic się nie stało, to znaczy czy nie uległ zmianie. Kod funkcji służącej do wyznaczenia kodu CRC wygląda następująco:

```
1    public String crcCalculate(String series) {
2           String crcCode="";
3           int REDIX=10;
4           for(int i=0;i<crcLength;i++) {
5                 series = series + "0";
6           }
7           char[] seriesChars = series.toCharArray();
8           for(int i=0;i<seriesChars.length-crcLength;i++) {
9                 char sign = seriesChars[i];
10                if(Character.toString(sign).equals("1")) {
11                      int control = i;
12                      for(int j=0;j<key.length();j++) {
13                            int                         signInt                         =
                              Character.getNumericValue(seriesChars[control]);
14                            int keyInt = Character.getNumericValue(key.charAt(j));
15                            seriesChars[control]                                        =
                              Character.forDigit((signInt^keyInt),REDIX);
16                            control++;
17                      }
18                }
19          }
20          for(int k=seriesChars.length-crcLength;k<seriesChars.length;k++) {
21                crcCode = crcCode + Character.toString(seriesChars[k]);
22          }
23          return crcCode;
24    }
```

Funkcja „crcCalculate” z klasy „Crc”

Funkcja ta działa następująco: najpierw w pętli for (linia 4 do 6) następuje dopisanie ciągu 0 długości o jeden mniejszej niż długość klucza. Następnie w linii 7 następuje zmiana formatu ciągu ze string na tablicę char by umożliwić dostęp do pojedynczego znaku i by móc go edytować. Następnie od linii 8 do linii 19 następują kroki wcześniej opisanego algorytmu od 2 do 6. W linii 10 następuje sprawdzenie czy najstarszy bit klucza będzie porównywany z 1, jeśli tak to następuje operacja XOR (symbol ^) pomiędzy kluczem i bitami z ciągu, a jej wynik jest zapisywany w miejscu bitów z ciągu. Na sam koniec, to znaczy w liniach 20-22 z otrzymanego ciągu odczytujemy kod CRC, czyli n-1 ostatnich bitów przy założeniu że klucz ma n bitów. Funkcja zwraca kod CRC.

Kolejną metodą jaką zawiera klasa `Crc` jest klasa sprawdzająca czy ciąg zawierający kod CRC wyznaczony przy pomocy metody „`crcCalculate`” jest poprawny, kod tej funkcji wygląda następująco:

```
1     public boolean crcCheck(String series) {
2           int REDIX=10;
3           char[] seriesChars = series.toCharArray();
4           for(int i=0;i<seriesChars.length-crcLength;i++) {
5                 char sign = seriesChars[i];
6                 if(Character.toString(sign).equals("1")) {
7                       int control = i;
8                       for(int j=0;j<key.length();j++) {
9                             int                         signInt                         =
                              Character.getNumericValue(seriesChars[control]);
10                            int keyInt = Character.getNumericValue(key.charAt(j));
11                            seriesChars[control]                                        =
                              Character.forDigit((signInt^keyInt),REDIX);
12                            control++;
13                      }
14                }
15          }
16          for(int k=0;k<seriesChars.length-crcLength;k++) {
17                char sign = seriesChars[k];
18                if(Character.toString(sign).equals("1")) {
19                      return false;
20                }
21          }
22          return true;
23    }
```

Funkcja „crcCheck" klasy „Crc"

Funkcja ta jest bardzo podobna do poprzedniej, pętla for zawierająca się w linijkach 4-15 jest identyczna co w poprzedniej funkcji, różnica polega na tym że na wejściu otrzymujemy ciąg zawierający kod CRC. W liniach 16-21 następuje sprawdzenie czy ciąg składa się z samych zer, jeśli nie oznacza to że ciąg jest inny niż ten utworzony przy ramkowaniu.

Procedura ramkowania wykorzystuję metodę „`crcCalculate`" w swoim przebiegu i wygląda ona nastpująco:

```
1     public String createSeries() {
2           String newSeries = frame;
3           int counter = 0;
4           for(int i=0;i<series.length();i++) {
5                 char sign = series.charAt(i);
6                 newSeries = newSeries + Character.toString(sign);
7                 if(Character.toString(sign).equals("1")) {
8                       counter++;
9                 }
10                else {
11                      counter = 0;
12                }
13                if(counter==5) {
14                      newSeries = newSeries + "0";
15                      counter = 0;
16                }
17          }
18          newSeries = newSeries + frame;
19          newSeries = newSeries + crc.crcCalculate(newSeries);
20          return newSeries;
21    }
```

Funkcja „createSeries" programu „Framing"

Funkcja ta przy użyciu zmiennych globalnych tworzy do zmiennej `newSeries` nowy ciąg na podstawie tego z pliku, na początku zostanie dopisany ciąg ze zmiennej `frame`, następnie w pętli for (linie 4-17) po każdym ciągu pięciu „1" pod rząd zostanie wstawione „0". Na koniec zostają dwie operacje, w linii 18 zostaje dodany znów ciąg ze zmiennej `frame`, a w linii 19 zostaje dodany kod CRC wyznaczony przy pomocy funkcji „`crcCalculate`". Na sam koniec w linii 20 zostanie zwrócony nowy ciąg zawierający już kod CRC.

Program ten posiada również funkcję mogącą odkodować ciąg stworzony przy użyciu funkcji „createSeries” przy tym samym kluczu, funkcja ta wygląda następująco

```
1     public String uncodeSeries() {
2           if(!crc.crcCheck(series)) {
3                 return "Blad przesylu";
4           }
5           String newSeries = "";
6           int counter = 0;
7           for(int i=8;i<series.length()-(7+crcKey.length());i++) {
8                 char sign = series.charAt(i);
9                 newSeries = newSeries + Character.toString(sign);
10                if(Character.toString(sign).equals("1")) {
11                      counter++;
12                }
13                else {
14                      counter = 0;
15                }
16                if(counter==5) {
17                      i++;
18                      counter = 0;
19                }
20          }
21          return newSeries;
22    }
```

Funkcja „uncodeSeries” programu „Framing”

Funkcja ta na początku (linie 2-4) sprawdza czy kod jest poprawny metodą „crcCheck”, jeśli tak to następuje jego odkodowanie, które polega na stworzeniu nowego ciągu do zmiennej newSeries. Ciąg zostaje „wydobyty” z ciągu wejściowego, to znaczy na początku i na końcu zostaje ominięta odpowiednia ilość znaków, następnie po każdym podciągu pięciu „1” zostaje pominięty znak, nie sprawdzamy czy jest to „0” gdyż cały ciąg został sprawdzony na samym początku funkcji. Tak „wydobyty” ciąg zostaje zapisany do zmiennej i zwrócony jako wynik wykonania funkcji.

2.1.2. Przykładowe wywołanie

Na początku musimy ustalić kilka zmiennych, to znaczy ciąg który będzie dodawany na początku i na końcu, który jest przechowywany w zmiennej globalnej frame, niech to będzie „01111110”. Niech klucz CRC, który również jest przechowywany w zmiennej globalnej wygląda następująco „1101”. Niech plik z w którym będzie ciąg nosi nazwę „ramka.txt”, a jego zawartość niech wygląda następująco: „0111110101000011”. (przykład jest prosty w celu ułatwienia prześledzenia zmian w ciągu). Po uruchomieniu kolejno metod: „getFile”, „createSeries” oraz „createFile” o trzymamy plik z nadaną nazwą „ramkaPo.txt”, jego zawartość wygląda następująco: „01111110011111001010000110111111000 01”. Łatwo można podzielić sobie otrzymany kod tak by zauważyć elementy dodane: „01111110 01111100101000011 01111110 001”. Elementy podkreślone zostały dodane przez algorytm, pierwszy i trzeci pochodzą ze zmiennej frame, ostatni to kod CRC, a podkreślone zero zostało dodane gdyż przed nim występuje ciąg pięciu jedynek. Jeśli teraz dla odpowiednich parametrów wywołamy funkcję: „getFile”, „uncodeSeries” oraz „createFile” to otrzymamy plik o nazwie „ramkaCoy.txt” w którym znajduje się ciąg: „0111110101000011”. Jest to taki sam ciąg jaki został podany do pliku „ramka.txt”, czyli to ten „oryginalny” ciąg.

2.2.Symulowanie ethernetowej metody dostępu do medium transmisyjnego

2.2.1. Opis programu

Program został napisany w języku Java. Jego funkcjonalność ogranicza się do określania czy w pojedynczym interwale (pojedynczym przebiegu pętli) „urządzenie” wysyła sygnał i przesłanie tego sygnału po kablu. Przyjmujemy że kablem jest wyzerowana tablica, a sygnałem ciąg znaków przesuwający się po nim. Przyjmujemy że urządzenia są podłączone na początku i na końcu tablicy. Funkcja służąca określeniu czy dane źródło wyśle sygnał wygląda następująco:

```
1     public boolean willSend(int source) {
2           Random generator = new Random();
3           double rand;
4           if(source==0) {
5                 rand = generator.nextDouble();
6                 if(rand>probability1st) {
7                       return false;
8                 }
9                 else {
10                      return true;
11                }
12          }
13          else if(source==cable.length-1) {
14                rand = generator.nextDouble();
15                if(rand>probability2nd) {
16                      return false;
17                }
18                else {
19                      return true;
20                }
21          }
22          return false;
23    }
```

Funkcja „willSend" programu „Tramsmission"

Funkcja ta korzysta z biblioteki `Random` do wygenerowania losowej liczby i jeśli dana liczba jest mniejsza, bądź równa prawdopodobieństwu wysłania dla danego źródła to funkcja zwraca `true` w przeciwnym wypadku zwracane jest `false`. Funkcja ta jest przystosowana dla dwóch źródeł, które są „podpięte" do tablicy w miejscach odpowiadających indeksom 0 i ostatniemu indeksowi.

Kolejną z funkcji programu jest symulowanie przesyłania sygnały po tablicy oraz wykrywanie ewentualnych kolizji które mogą wystąpić gdy dwa różne źródła będą próbowały wysłać sygnał w tym samym lub zbliżonym czasie, funkcja za to odpowiadająca wygląda następująco:

```
1     public boolean simulate() {
2           int position1st=0, send1st=0, toSend1st=0;
3           int position2nd=0, send2nd=0, toSend2nd=0;
4           int c;
5           for(int i=0;i<intervalNumber;i++) {
6                 if(send1st!=0 && send2nd!=0) {
7                       return false;
8                 }
9                 if(send1st==0) {
10                      if(willSend(0)) {
11                            send1st = cable.length*3;
12                            position1st=0;
13                            toSend1st=statementLength;
14                      }
15                }
16                else {
17                      if(position1st<cable.length && toSend1st>0) {
18                            cable[cable.length-1] = 1;
19                            for(int j=cable.length-1;j>0;j--) {
20                                  c = cable[j];
21                                  cable[j] = cable[j-1];
22                                  cable[j-1] = c;
23                            }
24                            toSend1st--;
25                            position1st++;
26                      }
27                      else if(position1st<cable.length && toSend1st==0) {
28                            for(int j=cable.length-1;j>0;j--) {
29                                  c = cable[j];
```

```
30                                  cable[j] = cable[j-1];
31                                  cable[j-1] = c;
32                            }
33                            position1st++;
34                      }
35                      else if(position1st==cable.length && toSend1st>=0) {
36                            if(toSend1st<=cable.length && toSend1st>0) {
37                                  toSend1st--;
38                            }
39                            else if(toSend1st==0) {
40                                  cable[cable.length-1] = 0;
41                                  for(int j=cable.length-1;j>0;j--) {
42                                        c = cable[j];
43                                        cable[j] = cable[j-1];
44                                        cable[j-1] = c;
45                                  }
46                            }
47                      }
48                      send1st--;
49                }
50          // w drugą stronę
51          }
52          return true;
53    }
```

Funkcja „simulate” program „Transmission”

Funkcja w liniach 2-4 ma zdefiniowane zmienne pomocnicze, odpowiednio w linii 2 dla symulacji w stronę rosnącą po indeksach, w linii 3 w stronę przeciwną. Jedna próba to kilka interwałów gdzie jeden interwał to pojedyncze przejście pętli. Na samym początku pętli sprawdzane jest czy oba urządzenia nie próbują wysyłać sygnału, jeśli tak to funkcja zwraca `false` gdyż dojdzie prędzej czy później do kolizji sygnałów. Jeśli do kolizji nie dochodzi to jest prowadzona procedura najpierw dla pierwszego urządzenia, a potem dla drugiego. Procedura ta dla pierwszego urządzenia zawiera się w liniach 9-49, kodu procedury dla drugiego urządzenia nie ma powyżej gdyż wygląda ona prawie tak samo tylko odbywa się w przeciwnym kierunku. Przy tej procedurze są wykorzystywane trzy zmienne pomocnicze, są to `position1st`, `send1st` oraz `toSensd1st`. Pierwsza przechowuje najdalej położony element w tablicy od źródła, druga ilość potrzebnych interwałów do zakończenia przesyłu, trzecia odpowiada za to ile elementów pozostało do przesłania. Tak więc w linii 9 następuje sprawdzenie wartości zmiennej `send1st`, gdy jest równa 0 wywołana zostaje funkcja `willSend` z parametrem 0 i gdy zwróci ona `true` oznacza to że od następnego interwału pierwsze urządzenie rozpocznie wysyłanie sygnału. Gdy zmienna ta ma wartość większą od zera oznacza to że jest w trakcie wysyłania sygnału, wtedy rozważane są trzy przypadki. Pierwszy (linie 17-26) jest to przypadek w którym element najbardziej oddalony z wysyłanego ciągu w tablicy nie jest jeszcze na ostatnim miejscu tablicy wtedy w pojedynczym interwale cały ciąg zostaje przesunięty o jedną pozycje i zostaje dopisany kolejny element ciągu. Drugi przypadek (linie 27-34) to taki gdy najdalszy element w tablicy znajduje się na ostatnim miejscu w tablicy i nie ma elementów do wysłania, wtedy jest zdejmowany element znajdujący się na ostatnim miejscu w tablicy, a ciąg zostaje przesunięty. Ostatni, trzeci przypadek (linie 35-47) ma miejsce gdy ostatni element znajduje się na ostatnim miejscu tablicy i są jeszcze elementy które oczekują na wysłanie.

2.2.2. Wywołanie programu dla określonych danych

Wywołajmy program dla ustalonych parametrów takich jak:

Długoś tablicy	20
Długość sygnału	40
Prawdopodobieństwo wysłania sygnału w pojedynczym interwale	0.05
Ilość interwałów w jednej próbie	80

Po przeprowadzeniu 10000 prób otrzymujemy prawdopodobieństwo kolizji, wywołanie programu wygląda następująco:

```
Prawdopodobieństwo wystąpienia kolizji:0.9521
```

Wywołanie 1

Po przeanalizowaniu wyniku dla tych danych łatwo zauważyć że prawdopodobieństwo wystąpienia kolizji w próbie zawierającej 80 interwałów jest bliskie 1 dla takich danych. Jest to dość intuicyjne gdy popatrzymy na to że jeden interwał odpowiada bardzo małemu odcinkowi czasu. Rozważmy sytuację której to odpowiada, jeden interwał dla tego przypadku to jest czas w którym wysyłany sygnał pokonuje 1/20 odległości, a w każdym takim odstępie czasu ma miejsce próba wysłania sygnału z zadanym prawdopodobieństwem. Co mimo pozornie nie dużego prawdopodobieństwa wysyłu daje bardzo dużą liczbę prób wysłania. Przeprowadźmy teraz próbę dla prawdopodobieństwa wysyłu o połowę mniejszego. Dane wejściowe będą teraz wyglądały następująco:

Długoś tablicy	20
Długość sygnału	40
Prawdopodobieństwo wysłania sygnału w pojedynczym interwale	0.025
Ilość interwałów w jednej próbie	80

Wywołanie programu daje następujący rezultat:

```
Prawdopodobieństwo wystąpienia kolizji:0.7268
```

Wywołanie 2

Jak widać prawdopodobieństwo wystąpienia kolizji w kabli zmalało lecz ciągle jest ono wysokie, zmiejszmy więc prawdopodobieństwo wysyłu bardziej, dane uruchomienia:

Długoś tablicy	20
Długość sygnału	40
Prawdopodobieństwo wysłania sygnału w pojedynczym interwale	0.002
Ilość interwałów w jednej próbie	80

```
Prawdopodobieństwo wystąpienia kolizji:0.0189
```

Wywołanie 3

Jak widać wyniki drastycznie spadły, możemy przyjąć że są to wyniki zadowalające. Teraz możemy się zastanowić nad wpływem długości kabla na wyniki. Zwiększmy ją dwukrotnie.

Długoś tablicy	40
Długość sygnału	40
Prawdopodobieństwo wysłania sygnału w pojedynczym interwale	0.002
Ilość interwałów w jednej próbie	80

```
Prawdopodobieństwo wystąpienia kolizji:0.0187
```

Wywołanie 4

Jak widać długość tablicy (kabla) nie ma żadnego lub bardzo mały wpływ na prawdopodobieństwo kolizji sygnałów.

3. Wnioski

Program do ramkowania działa poprawnie, warto zaznaczyć że trzeba trzymać się pewnych założeń, czy też standardów by można było przeprowadzać tę operację w obie strony lub by można było stwierdzić czy podczas przesyłania danych

nie uległy one uszkodzeniu. Oczywiście metoda przedstawiona w programie nie daje 100% pewności że takowy błąd zostanie wykryty, lecz istnieje na to duża szansa.

Symulacja dostępu do medium transmisyjnego pokazała że największy wpływ na kolizje mają urządzenia wysyłające sygnały, a zatem również użytkownicy. Czym mniej sygnałów tym mniej kolizji, logiczne. Próby pokazały również że długość kabla nie mają na to wpływu. Intuicja podpowiada jednak że odpowiednia zależność pomiędzy długością sygnału, a długością kabla może spowodować zmniejszenie prawdopodobieństwo wystąpienia kolizji.