09.05.2020

Lista 3

Technologie sieciowe

Dominika Szydło

Lista 3

Technologie sieciowe

Zadanie 1

W zadaniu pierwszym należało napisać program potrafiący zakodować i odkodować wiadomość za pomocą metody „rozpychania" bitów i CRC. Kodowanie odbywa się zgodnie z następującą procedurą:

1. Obliczenie 32-bitowego CRC wiadomości. Jeżeli liczba ta jest mniejsza od 2^{31}, to należy ją uzupełnić od początku zerami tak, by faktycznie miała długość 32 bitów.
2. Dopisanie sformatowanego CRC na początek wiadomości.
3. Wstawienie znaku '0' po każdym napotkanym ciągu pięciu jedynek w wiadomości.
4. Wstawienie na początek i koniec wiadomości ciągu znaków '01111110'

Dla ciągu znaków 10111011111000011111100011011000 procedura daje wynik 01111110111001000001101000111101101110001011101111100000111110100011011000001111110, gdzie kolorem niebieskim zaznaczone są znaki dodane w kroku czwartym, pomarańczowym dodane w kroku trzecim, a różowym dodane w kroku 2.

Odkodowanie polega na odwróceniu listy kroków kodowania, tj.:

1. Usunięcie z początku i końca wiadomości ciągu znaków '01111110'.
2. Usunięcie znaku '0' po każdym napotkanym ciągu pięciu jedynek.
3. Podzielenie wiadomości na dwie części – od pierwszego do trzydziestego drugiego bitu i od trzydziestego trzeciego bitu do końca. Pierwsza otrzymana część to CRC wiadomości, druga to zakodowane dane.
4. Obliczenie CRC danych – jeżeli nie zgadza się z tym otrzymanym, to wiadomość nie została zakodowana poprawnie. W przeciwnym wypadku wszystko jest w porządku i otrzymano poprawne dane.

W pliku zad1.py znajdują się implementacje powyższych procedur. W funkcji main() program generuje pseudolosowy ciąg znaków '0' i '1' o długości 160 i zapisuje go do pliku z.txt. Następnie odczytuje ten ciąg z pliku, koduje go i zapisuje do pliku w.txt. Na koniec odczytuje zakodowany ciąg z pliku w.txt, odkodowuje go i wypisuje na ekran. To co wypisuje jest jednocześnie zawartością pliku z.txt.

Zadanie 2

W zadaniu drugim należało napisać program symulujący metodę dostępu do medium transmisyjnego CSMA/CD. Zadanie wykonałam implementując klasę Network reprezentującą kanał komunikacyjny i funkcję transmit(), którą wykonują wątki reprezentujące transmitujących użytkowników. Klasa Network posiada atrybuty:

- channel – tablica o długości 50 reprezentująca kanał. Jeżeli dana komórka nie przechowuje transmitowanych danych to jej zawartość to znak '-'.
- collision_detected – zmienna boolowska przechowująca informację o tym czy na kanale wykryto kolizję.
- transmitting - zmienna boolowska przechowująca informację o tym czy ktoś wykonuje transmisję na kanale.
- lock – lock blokujący operacje na kanale.

Funkcja transmit() natomiast przyjmuje jako argument wartości:

- user_id – ID użytkownika nadane mu w funkcji main(), używane do identyfikacji działań w logach wypisywanych przez program.
- index – lokalizacja użytkownika w kanale transmisyjnym.
- t – podana przez użytkownika jako parametr programu liczba sekund przez jaką ma działać główna pętla funkcji.

- msg_count – pseudolosowo wygenerowana maksymalna liczba wiadomości jaką może wysłać dany wątek.
- color – przydzielony w funkcji main() kolor w jakim będą wypisywane logi aktywności danego użytkownika.

Główna pętla funkcji tranmit() przebiega (w pewnym uproszczeniu) według następujących kroków:

1. Generowanie pseudolosowej wartości ze zbioru {0, 1} – jeżeli wylosowano 1 i wątek nie wykorzystał limitu wiadomości do wysłania to przechodzi do następnego kroku, jeżeli nie to do następnego obiegu pętli.
2. Generowanie 8-bitowej wiadomości do transmisji.
3. Ustawienie liczby podejść do przesłania danej wiadomości na 1.
4. Dopóki liczba podejść jest mniejsza niż 4 wykonywane są kroki od 5. do 9.
5. Sprawdzenie czy na kanale doszło do kolizji – jeżeli tak, to zwiększenie liczby podejść o jeden, sleep przez dwie sekundy i przejście do następnego obiegu pętli
6. Sprawdzenie czy ktoś dokonuje transmisji na kanale – jeżeli tak, to zwiększenie liczby podejść o 1 i przejście do następnego obiegu pętli.
7. Próba transmisji – na początku sleep przez 0,001 sekundy, a dopiero potem zmiana statusu kanału na zajęty. Sleep występuje, aby można było zaobserwować kolizję, co przez naturę wątków bez niego byłoby dosyć trudne.
8. Dopóki wiadomość nie dojdzie do końców kanału kolejne bity są propagowane w dwóch kierunkach z lokalizacji użytkownika. Jeżeli w procesie zajdzie kolizja to wykonywany jest sleep przez dwie sekundy, kanał jest czyszczony i ustawiony na wolny, inkrementowana zostaje liczba podejść i następuje przejście do kolejnego obiegu pętli.
9. Jeżeli wiadomość udało się poprawnie przesłać to kanał zostaje ustawiony na wolny, liczba wysłanych wiadomości inkrementowana, a liczba podejść ustawiona na 4, żeby program wyszedł z pętli i wrócił do wykonywania pętli głównej.

Rozwiązanie zadania znajduje się w pliku zad2.py, któremu jako argument należy podać liczbę sekund przez jaką ma się wykonywać główna pętla każdego wątku, wartość zalecana przeze mnie to 2-4, ponieważ otrzymane w tym czasie logi pozwalają na zorientowanie się co się dzieje w symulacji i nie jest ich na dużo.