Zadanie programistyczne nr 2
z Sieci komputerowych

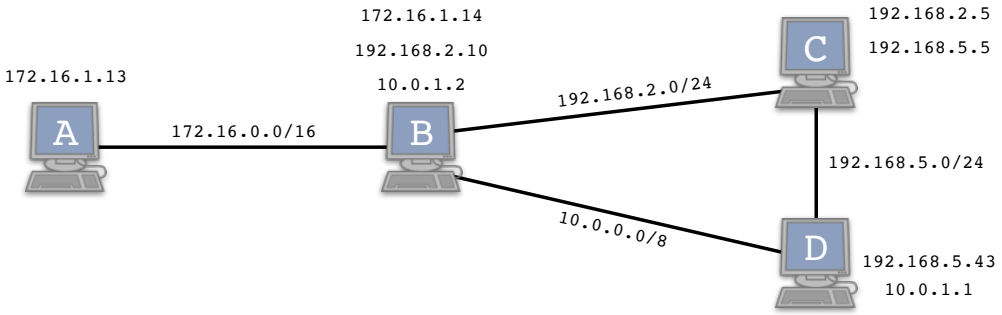
1 Opis zadania

Napisz program `router`, który będzie implementować algorytm wektora odległości i tworzyć tablice przekazywania. Twój program zostanie uruchomiony na wielu komputerach (po jednej instancji na jednym komputerze). Instancja programu powinna wczytać konfigurację ze standardowego wejścia; konfiguracja będzie zawierać adresy IP komputera, na którym działa instancja oraz adresy podłączonych do niego bezpośrednio sieci. Komunikacja pomiędzy instancjami powinna odbywać się za pośrednictwem pakietów UDP wysyłanych na adres rozgłoszeniowy danej sieci. Zaprojektowanie formatu wysyłanych wiadomości jest częścią zadania.

Twój program nie ma przeprowadzać interakcji z istniejącą na komputerze tablicą routingu czy przekazywania, tj. odczytywać z niej danych ani do niej zapisywać. Wyznaczoną tablicę przekazywania program powinien wyświetlać co pewien czas na ekranie. Twój program musi być przygotowany na sytuacje, że wysłanie pakietu nie powiedzie się, tj. funkcja `sendto` zwróci błąd lub wysłany pakiet nie dotrze do odbiorcy.

1.1 Przykładowa sieć i format konfiguracji

Przykładową konfigurację złożoną z 4 sieci i 4 komputerów przedstawiono na poniższym rysunku.



Komputer `D` na standardowym wejściu otrzyma następującą konfigurację

```
2
10.0.1.1 netmask /8 distance 3
192.168.5.43 netmask /24 distance 2
```

Pierwszy wiersz konfiguracji zawiera liczbę naturalną n będącą liczbą adresów IP komputera i zarazem liczbą podłączonych bezpośrednio sieci. Każdy z kolejnych n wierszy zawiera następujące informacje rozdzielone pojedynczymi odstępami:

- adres IP w notacji dziesiętnej rozdzielonej kropkami;
- napis `netmask`
- maskę sieci w formacie `/xx`
- napis `distance`
- odległość do tej sieci (do dowolnego komputera w tej sieci).

1.2 Działanie programu

Każda instancja programu `router` powinna implementować następujące cechy.

1. Co pewien zdefiniowany w programie czas (10-30 sekund, nazywany *turą*) powinna wysyłać cały swój aktualny wektor odległości (pary składające się z sieci i odległości do nich) na adresy rozgłoszeniowe wszystkich sąsiadujących z nią sieci. Dane powinny być wysyłane w jednym lub większej liczbie pakietów UDP.
2. Nasłuchiwać na ustalonym porcie i odbierać rozsyłane wektory odległości.
3. Utrzymywać bieżący wektor odległości i tablicę przekazywania, w której trasy obliczane są jako najkrótsze znane ścieżki do danych sieci. Tablica powinna być wyświetlana co turę na przykład w poniższym formacie:

   ```
   10.0.0.0/8 unreachable
   192.168.5.0/24 distance 2 connected directly
   192.168.2.0/24 distance 4 via 192.168.5.5
   172.16.0.0/16 distance 6 via 192.168.5.5
   ```

4. Reagować na nieosiągalność sąsiadów:
   (a) Błąd zwrócony przez funkcję `sendto` powinien skutkować wpisaniem do wektora odległości nieskończonej do sąsiadującej sieci.
   (b) Brak wiadomości przez kilka tur od komputera (znajdującego się w sąsiedniej sieci), który w tablicy przekazywania jest pierwszym komputerem na trasie do sieci A powinien skutkować wpisaniem w wektorze odległości do sieci A odległości nieskończonej.
5. Rozgłaszać wpisy wektora z odległościami nieskończonymi przez kilka kolejnych tur (w takim samym trybie jak inne wpisy), a następnie usuwać je z wektora odległości i tablicy przekazywania.
6. Reagować poprawnie jeśli osiągalność sąsiadującej bezpośrednio sieci lub sąsiedniego komputera zostanie przywrócona (np. jeśli otrzymamy wektor odległości od innego komputera z tej sieci).

Twój program nie musi implementować dodatkowych rozszerzeń algorytmu wektora odległości takich jak przyspieszone uaktualnienia czy zatruwanie wsteczne. Twoja specyfikacja powinna natomiast definiować (niezbyt dużą) wartość graniczną, powyżej której odległość w sieci jest uznawana za nieskończoną, co umożliwi rozwiązywanie problemu zliczania do nieskończoności.

1.3 Specyfikacja komunikacji

W osobnym pliku tekstowym `spec.txt` opisz szczegóły komunikacji. W szczególności powinna znaleźć się tam odpowiedź na następujące pytania:

- Na jakim porcie nasłuchuje instancja programu `router`?
- Jaki jest format przesyłanych wiadomości? Ile elementów wektora odległości może maksymalnie zmieścić się w pakiecie?
- Jaka jest długość tury w sekundach? Po jakim czasie trasy zostają oznaczone jako nieosiągalne, a po jakim czasie nieosiągalne trasy będą wyrzucane z tablicy?
- Jaką odległość uznajemy za nieskończoną?

Uwaga: ten plik może być wspólny ale dla maksymalnie trzech osób (ich nazwiska powinny się pojawić w tym pliku). W takim przypadku testowanie powinno obejmować uruchamianie w jednej sieci programów różnych autorów.

2 Uwagi techniczne

Pliki Swojemu ćwiczeniowcowi należy dostarczyć jeden spakowany plik zawierający katalog z programem. Katalog powinien zawierać:

- Kod źródłowy w C lub C++, czyli pliki `*.c` i `*.h` lub pliki `*.cpp` i `*.h`. Każdy plik `*.c` i `*.cpp` na początku powinien zawierać w komentarzu imię, nazwisko i numer indeksu autora.
- Plik `Makefile` pozwalający na kompilację programu po uruchomieniu `make`.
- Plik `spec.txt` zawierający szczegółowy opis komunikacji.
- Ewentualnie plik `README`.

W katalogu tym **nie** powinno być żadnych innych plików, w szczególności skompilowanego programu, obiektów `*.o`, czy plików źródłowych nie należących do projektu.

Kompilacja Kompilacja i uruchamianie przeprowadzane zostaną w 64-bitowym środowisku Linux. Kompilacja w przypadku C ma wykorzystywać standard ISO C99 z ewentualnymi rozszerzeniami GNU (opcja kompilatora `-std=c99` lub `-std=gnu99`). Kompilacja powinna wykorzystywać opcje `-Wall` i `-W`. Podczas kompilacji nie powinny pojawiać się ostrzeżenia.

3 Sposób oceniania programów

Poniższe uwagi służą ujednoliceniu oceniania w poszczególnych grupach. Napisane są jako polecenia dla ćwiczeniowców, ale studenci powinni **koniecznie się** z nimi zapoznać, gdyż będziemy się ściśle trzymać poniższych wytycznych. Programy będą testowane na zajęciach w obecności autora programu. Na początku program uruchamiany jest w różnych warunkach i otrzymuje za te uruchomienia od 0 do 10 punktów. Następnie obliczane są ewentualne punkty karne. Oceniamy z dokładnością do 0,5 punktu. Jeśli ostateczna liczba punktów wyjdzie ujemna wstawiamy zero. (Ostatnia uwaga nie dotyczy przypadków plagiatów lub niesamodzielnych programów).

Testowanie: punkty dodatnie Rozpocząć od kompilacji programu. W przypadku programu niekompilującego się, stawiamy 0 punktów, nawet jeśli program będzie ładnie wyglądał.

2 pkt. Skonfigurować cztery komputery połączone ze sobą w kwadrat. Powinny być zdefiniowane cztery sieci (z różnymi maskami niekoniecznie będącymi wielokrotnościami liczby 8). Każdy komputer powinien być podłączony do dwóch z sieci (odpowiednio kartami `eth0` i `eth1`) i mieć dwa adresy IP, po jednym z każdej sieci.

Stworzyć cztery pliki konfiguracyjne na tych komputerach i podać je na wejściu czterem instancjom programu `router`. Obejrzeć wyświetlane przez komputery komunikaty. Po pewnym czasie każdy z komputerów powinien znać trasę do wszystkich czterech sieci i tablica przekazywania powinna się już nie zmieniać.

3 pkt. Wyłączyć jedną z czterech instancji `router`. Po pewnym czasie całość powinna zbiegnąć do stanu, w którym nie ma tras prowadzących przez ten router. Wszystkie sieci powinny pozostać osiągalne. Następnie należy włączyć tę instancję ponownie. Po pewnym czasie całość powinna zbiegnąć do stanu wykorzystującego ten router (tam gdzie jest on na najkrótszej trasie do danej sieci).

3 pkt. Wykonać poprzedni punkt, ale wyłączając jeden z ośmiu interfejsów poleceniem `ifconfig eth`*x* `down`. Po pewnym czasie nie powinno być trasy prowadzącej przez wyłączoną kartę sieciową. Wszystkie sieci powinny pozostać osiągalne. Następnie należy wyłączyć drugi interfejs prowadzący do tej sieci. Po pewnum czasie sieć powinna być nieosiągalna dla wszystkich komputerów. Na końcu należy włączyć oba interfejsy; całość powinna zbiegnąć do stanu wykorzystującego obie karty sieciowe.

2 pkt. Te punkty można są przewidziane za specyfikację programu w pliku `spec.txt`. Na podstawie specyfikacji i niniejszego tekstu powinno być możliwe stworzenie programu, który komunikuje się z Twoim. Punkty te można otrzymać tylko, jeśli program uzyskał co najmniej 3 punkty za poprzednie podpunkty.

Punkty karne Punkty karne przewidziane są za następujące usterki.

do -3 pkt. Zła / nieczytelna struktura programu: wszystko w jednym pliku, brak modularności i podziału na funkcjonalne części, w szczególności mieszanie funkcji które wysyłają pakiety z funkcjami które odbierają, mieszanie części wysyłającej pakiety z częścią wyświetlającą komunikaty dla użytkownika, niekonsekwentne wcięcia, powtórzenia kodu.

-1 pkt. Brak sprawdzania poprawności wywołania funkcji systemowych, takich jak `bind()` czy `sendto()`.

-1 pkt. Zły plik `Makefile` lub jego brak: program powinien się kompilować poleceniem `make`, polecenie `make clean` powinno czyścić program z tymczasowych obiektów (plików `*.o`), polecenie `make distclean` powinno usuwać skompilowane programy i zostawiać tylko pliki źródłowe.

-1 pkt. Niewłaściwa kompilacja: nietrzymanie się opcji podanych w zadaniu, ostrzeżenia wypisywane przy kompilacji, kompilacja bezpośrednio do pliku wykonywalnego bez tworzenia obiektów tymczasowych `k*.o`.

(-3/-6 pkt) Kara za wysłanie programu z opóźnieniem: -3 pkt. za opóźnienie do 1 tygodnia, -6 pkt. za opóźnienie do 2 tygodni. Programy wysyłane z większym opóźnieniem nie będą sprawdzane.

Marcin Bieńkowski