Zadanie programistyczne nr 2
z Sieci komputerowych

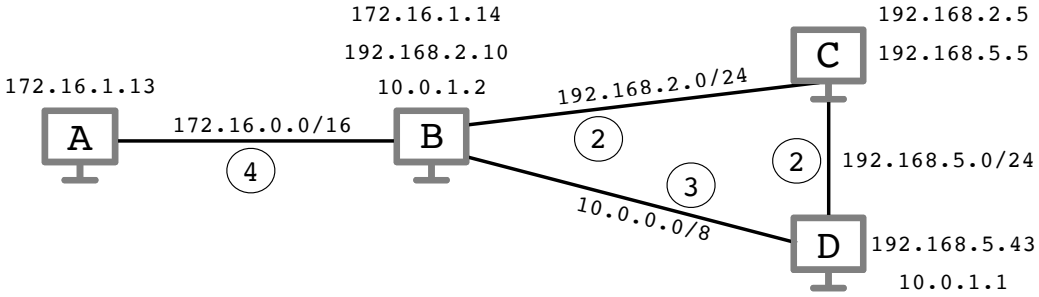
1 Opis zadania

Napisz program `router`, który będzie implementować algorytm wektora odległości i tworzyć tablice przekazywania. Twój program zostanie uruchomiony na wielu maszynach (po jednej instancji na jednej maszynie). Instancja programu powinna wczytać konfigurację ze standardowego wejścia; konfiguracja będzie zawierać adresy IP maszyny, na której działa instancja oraz adresy podłączonych do niej bezpośrednio sieci. Komunikacja pomiędzy instancjami powinna odbywać się za pośrednictwem datagramów UDP wysyłanych na adres rozgłoszeniowy danej sieci (zawierających odpowiednio sformatowane wpisy z tablicy routingu).

Twój program nie ma przeprowadzać interakcji z istniejącą na maszynie 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 datagramu nie powiedzie się, tj. funkcja `sendto()` zwróci błąd lub wysłany datagram nie dotrze do odbiorcy.

1.1 Przykład i format konfiguracji

Przykładowy zestaw złożony z 4 sieci i 4 maszyn przedstawiono na poniższym rysunku.



Liczby w kółkach oznaczają „odległości” do danych sieci od danej maszyny i są liczbami całkowitymi dodatnimi ustalanymi wyłącznie na potrzeby tego zadania. Przykładowo na powyższym rysunku odległość od maszyny `B` do dowolnego urządzenia w sieci `192.168.2.0/24` wynosi 2 (w szczególności taka jest odległość od maszyny `B` do maszyny `C`).

Każda instancja programu otrzyma na standardowym wejściu własną konfigurację, zależną od maszyny, na której została uruchomiona. Pierwszy wiersz konfiguracji zawiera liczbę naturalną n będącą liczbą interfejsów maszyny (i zarazem liczbą przypisanych do maszyny adresów IP i liczbą sieci, do których maszyna jest podłączona). Każdy z kolejnych n wierszy zawiera informację o jednym interfejsie w następującym formacie (poniższe trzy elementy są rozdzielone pojedynczymi odstępami):

- adres IP interfejsu w notacji CIDR: cztery liczby w postaci dziesiętnej rozdzielone kropkami, ukośnik i maska podsieci (liczba z zakresu $[0, 32]$),
- napis `distance`,
- odległość do tej sieci.

Możesz założyć, że konfiguracja będzie w powyższym formacie i nie musisz tego sprawdzać. Możesz założyć, że żadna para różnych sieci występujących w konfiguracjach nie ma wspólnych adresów IP.

Przykładowo instancja programu uruchomiona na maszynie `D` na standardowym wejściu otrzyma konfigurację

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

1.2 Działanie programu

Każda instancja programu `router` powinna implementować następujące cechy.

1. Nasłuchiwać na porcie 54321, odbierać i przetwarzać rozsyłane przez innych elementy wektora odległości.
2. Co pewien zdefiniowany w programie czas (5–20 sekund, nazywany *turą*) wysyłać wszystkie elementy swojego aktualnego wektora odległości (pary składające się z sieci i odległości do nich) do adresów rozgłoszeniowych wszystkich sąsiednich sieci. Każdy element wektora powinien być wysyłany w osobnym datagramie UDP, skierowanym do portu 54321, w formacie określonym w kolejnej sekcji.
3. Utrzymywać tablicę routingu, czyli bieżący wektor (najkrótszych) odległości do poszczególnych sieci, a przypadku sieci niebezpośrednio podłączonych również informację o pierwszym routerze na trasie do takiej sieci. Tablica ta powinna być wyświetlana na ekranie co pewien zdefiniowany w programie czas (nie rzadziej niż raz na turę). Przykładowo, jeśli wszystkie łącza będą działać, to po pewnym czasie maszyna `D` powinna wyświetlać tablicę routingu w formacie podobnym do poniższego:

   ```
   10.0.0.0/8 distance 3 connected directly
   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 7 via 10.0.1.2
   ```

   Jeśli następnie wyłaczymy interfejsy maszyn `B` i `D` prowadzące do sieci `10.0.0.0/8`, to po pewnym czasie wyświetlana tablica maszyny `D` powinna wyglądać następująco:

   ```
   10.0.0.0/8 unreachable connected directly
   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 8 via 192.168.5.5
   ```

4. Reagować na nieosiągalność sąsiednich routerów, uaktualniając odległość do nich na nieskończoną (a także do prowadzących przez nie sieci). O nieosiągalności dowiadujemy się z dwóch źródeł: (1) funkcja `sendto()` może zwrócić błąd (np. w przypadku nieaktywnego interfejsu), (2) nie dostaniemy od maszyny z sąsiedniej sieci przez parę tur żadnej wiadomości. Warto pamiętać, że w drugim przypadku bezpośrednio podłączona do nas sieć zawierająca ten router jest wciąż osiągalna.
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. Wyjątkiem są bezpośrednio połączone sieci: po kilku turach należy przestać rozgłaszać o nich informację, ale nie należy usuwać o nich informacji. W szczególności należy nadal próbować

wysyłać do takich sieci elementy wektora odległości i sensownie zareagować na sytuację, gdy po awarii łącze między dwoma maszynami zostanie naprawione.

Twój program nie musi implementować dodatkowych rozszerzeń algorytmu wektora odległości takich jak przyspieszone uaktualnienia czy zatruwanie ścieżki zwrotnej. 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 Format datagramu UDP

Każdy element wektora odległości należy wysyłać w osobnym datagramie UDP w następującym formacie. Dane datagramu powinny składać się z 9 bajtów.

- Bajty od 1 do 5 opisują sieć. Pierwsze cztery z tych pięciu bajtów to adres IP, zaś piąty to długość maski sieci (liczba z zakresu $[0, 32]$). Adres IP powinien być zapisany w sieciowej kolejności bajtów (najpierw najbardziej istotny).
- Bajty od 6 do 9 zawierają nieujemną liczbę całkowitą określającą odległość do tej sieci. Wartość $2^{32} - 1$ oznacza odległość nieskończoną. Liczba powinna zostać zapisana w sieciowej kolejności bajtów.

Możesz założyć, że jeśli Twój program otrzymuje datagram UDP, to jego format spełnia powyższe warunki i nie trzeba tego sprawdzać.[1]

2 Uwagi techniczne

Ta sekcja jest taka sama dla wszystkich zadań programistycznych.

Pliki. Sposób utworzenia napisu oznaczanego poniżej jako *imie_nazwisko*: Swoje (pierwsze) imię oraz nazwisko zapisz wyłącznie małymi literami, zastępując litery ze znakami diakrytycznymi przez ich łacińskie odpowiedniki (usuń ogonki). Pomiędzy imię i nazwisko wstaw znak podkreślenia.

Prowadzącemu ćwiczeniopracownię należy dostarczyć plik *imie_nazwisko*`.tar.xz` z archiwum (w formacie `tar`, spakowane programem `xz`) zawierającym pojedynczy katalog o nazwie *imie_nazwisko* z następującymi plikami.

- Kod źródłowy w C, C++ lub Rust, czyli pliki `*.c`, `*.h`, `*.cpp`, `*.h` lub `*.rs`. Każdy plik źródłowy na początku powinien zawierać w komentarzu imię, nazwisko i numer indeksu autora lub autorki.
- Plik `Makefile` pozwalający na kompilację programu po uruchomieniu `make`. Takie wymaganie obowiązuje również w przypadku kodów źródłowych w Rust (wywołanie `make` może uruchamiać `cargo` lub `rustc`).
- Ewentualnie plik `README.txt` lub `README.md`.

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

[1] W prawdziwych zastosowaniach byłby to bardzo zły pomysł.

Kompilacja. Kompilacja i uruchamianie przeprowadzone zostaną w 64-bitowym środowisku Linux; w szczególności kody powinno dać się uruchomić w maszynie wirtualnej Virbian bez instalacji dodatkowych pakietów.

- Kompilacja w przypadku C ma wykorzystywać standard C17 lub C18 z ewentualnymi rozszerzeniami GNU (opcja kompilatora `-std=c17`, `-std=gnu17`, `-std=c18` lub `-std=gnu18`).
- Kompilacja w przypadku C++ ma wykorzystywać standard C++14, C++17 lub C++20 z ewentualnymi rozszerzeniami GNU (opcja kompilatora `-std=c++14`, ..., `-std=c++20`, `-std=gnu++14`, ..., `-std=gnu++20`).
- Kompilacja w przypadku Rust ma wykorzystywać edycję 2018, 2021 lub 2024 (opcja kompilatora `-edition 2018`, ..., `-edition 2024`).

Kompilacja w przypadku C i C++ powinna korzystać z opcji `-Wall` i `-Wextra`. Podczas kompilacji w dowolnym języku nie powinny pojawiać się ostrzeżenia.

3 Sposób oceniania programów

Poniższe uwagi służą ujednoliceniu oceniania w poszczególnych grupach. Napisane są jako polecenia dla prowadzących, ale studenci powinni **koniecznie** się z nimi zapoznać, gdyż prowadzący będą się ich trzymać przy sprawdzaniu.

- Programy będą testowane na zajęciach w obecności autora lub autorki programu.
- Program niekompilujący się otrzymuje 0 punktów, nawet jeśli ładnie wygląda.
- Program uruchamiany jest w różnych warunkach opisanych poniżej i otrzymuje za te uruchomienia od 0 do 10 punktów.
- Następnie obliczane są ewentualne punkty ujemne zgodnie z listą podaną poniżej.
- 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 Należy wykonać następujące testy.

2 pkt. Skonfigurować cztery maszyny wirtualne połączone ze sobą w kwadrat. Powinny być zdefiniowane cztery sieci (z różnymi maskami niekoniecznie będącymi wielokrotnościami liczby 8). Każda maszyna powinna być podłączona do dwóch sieci i mieć dwa adresy IP, po jednym z każdej sieci.

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

4 pkt. Wyłączyć jedną z czterech instancji `router`. Po pewnym czasie całość powinna zbiegnąć do stanu bez tras prowadzących przez tę maszynę, lecz z osiągalnymi wszystkimi sieciami. Następnie należy włączyć tę instancję ponownie. Po pewnym czasie całość powinna zbiegnąć do stanu wykorzystującego tę maszynę (tam gdzie jest ona na najkrótszej trasie do danej sieci).

4 pkt. Wykonać poprzedni punkt, ale wyłączając jeden z ośmiu interfejsów poleceniem `ip link set down dev ...`. Po pewnym czasie nie powinno być trasy prowadzącej przez wyłączoną kartę sieciową. Docelowo wszystkie sieci powinny być osiągalne. Następnie należy wyłączyć drugi interfejs prowadzący do tej sieci. Po pewnym czasie sieć powinna być nieosiągalna dla wszystkich maszyn. Na końcu należy włączyć oba interfejsy; całość powinna zbiegnąć do stanu wykorzystującego obie karty sieciowe.

Testowanie: punkty ujemne. Punkty ujemne przewidziane są za następujące usterki.

do -3 pkt. Zła/nieczytelna struktura programu: brak modularności i podziału na funkcjonalne części, niekonsekwentne wcięcia, powtórzenia kodu.

-2 pkt. Aktywne czekanie zamiast zasypiania do momentu otrzymania pakietu.

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

-1 pkt. Nietrzymanie się specyfikacji wejścia i wyjścia. Przykładowo: wyświetlanie nadmiarowych informacji diagnostycznych lub inna niż w specyfikacji obsługa parametrów.

-1 pkt. Zły plik `Makefile` lub jego brak.

- Program powinien się kompilować poleceniem `make`.
- Poszczególne pliki `*.c` i `*.cpp` powinny kompilować się do obiektów tymczasowych `*.o` a następnie powinny być konsolidowane do wykonywalnego programu.
- Polecenie `make clean` powinno czyścić katalog z tymczasowych obiektów (plików `*.o`), zaś polecenie `make distclean` powinno usuwać te obiekty i wykonywalny program pozostawiając tylko pliki źródłowe.

-3 pkt. Kara za wysłanie programu po terminie; opóźnienie nie może być większe niż 1 tydzień.

Materiały do kursu znajdują się w systemie SKOS: https://skos.ii.uni.wroc.pl/.

Marcin Bieńkowski